Acceleration in optimization

Damien Scieur

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences Lettres
PSL Research University

Préparée à l'École normale supérieure

Acceleration in Optimization

Sur l'accélération des méthodes d'optimisation

École doctorale n°386

ÉCOLE DOCTORALE DE SCIENCES MATHÉMATIQUES DE PARIS CENTRE

Spécialité INFORMATIQUE

COMPOSITION DU JURY :

Antonin Chambolle
CMAP EP Paris, Président du jury

Alexandre d'Aspremont
Inria Paris, Directeur de thèse

Francis Bach
Inria Paris, Directeur de thèse

Joseph Salmon
TELECOM ParisTech, Membre du jury

Yurii Nesterov
UCL, Membre du jury

Soutenue par Damien Scieur
le 11 septembre 2018

Dirigée par
Alexandre d'Aspremont et Francis Bach

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

Mrglrglmrglmrrrlggg!

Murky the Murloc

Uuuuuuuuurrrr Aaaahhrrrrrrr,
Uuuhhhhrrrrrrrr, Urrh Aaarrrh.

Chewbacca

Quidquid latine dictum sit, altum
sonatur.

Loth d'Orcanie

Résumé

Dans de nombreux domaines, comme par exemple l'optimisation, la performance d'une méthode est souvent caractérisée par son taux de convergence. Cependant, accélérer un algorithme requiert une certaine connaissance de la structure du problème et de telles améliorations sont le fruit d'une étude au cas-par-cas. De nombreuses techniques d'accélération ont été développées ces dernières décennies et sont maintenant massivement utilisées. En dépit de leur simplicité, ces méthodes sont souvent basées sur des arguments purement algébriques et n'ont généralement pas d'explications intuitives.

Récemment, de nombreux travaux ont été menés pour faire des liens entre les algorithmes accélérés et d'autres domaines scientifiques, comme par exemple avec la théorie du contrôle ou des équations différentielles. Cependant, ces explications reposent souvent sur des arguments complexes et la plupart utilise des outils non-conventionnels dans leur analyse.

Une des contributions de cette thèse est une tentative d'explication des algorithmes accélérés en utilisant la théorie des méthodes d'intégration, qui a été très étudiée et jouit d'une analyse théorique solide. En particulier, nous montrons que les méthodes d'optimisation sont en réalité des instances de méthode d'intégration, lorsqu'on intègre l'équation du flot de gradient. Avec des arguments standards, nous expliquons intuitivement l'origine de l'accélération.

De l'autre côté, les méthodes accélérées ont besoin de paramètres supplémentaires, en comparaison d'autres méthodes plus lentes, qui sont généralement difficiles à estimer. De plus, ces schémas sont construits pour une configuration particulière et ne peuvent pas être utilisés autre part.

Ici, nous explorons une autre approche pour accélérer les algorithmes d'optimisation, qui utilise des arguments d'accélération générique. En analyse numérique, ces outils ont été développés pour accélérer des séquences de scalaires ou de vecteurs, en construisant parallèlement une autre séquence avec un meilleur taux de convergence. Ces méthodes peuvent être combinées avec un algorithme itératif, l'accélérant dans la plupart des cas. En pratique, ces méthodes d'extrapolation ne sont pas tellement utilisées, notamment dû à leur manque de garanties de convergence et leur instabilité. Nous étendons ces méthodes en les régularisant, ce qui permettra une analyse théorique plus profonde et des résultats de convergence plus fort, en particulier lorsqu'elles sont appliquées à des méthodes d'optimisation.

Abstract

In many different fields such as optimization, the performance of a method is often characterized by its rate of convergence. However, accelerating an algorithm requires a lot of knowledge about the problem's structure, and such improvement is done on a case-by-case basis. Many accelerated schemes have been developed in the past few decades and are massively used in practice. Despite their simplicity, such methods are usually based on purely algebraic arguments and often do not have an intuitive explanation.

Recently, heavy work has been done to link accelerated algorithms with other fields of science, such as control theory or differential equations. However, these explanations often rely on complex arguments, usually using non-conventional tools in their analysis.

One of the contributions of this thesis is a tentative explanation of optimization algorithms using the theory of integration methods, which has been well studied and enjoys a solid theoretical analysis. In particular, we will show that optimization scheme are special instance of integration methods when integrating the classical gradient flow. With standard arguments, we intuitively explain the origin of acceleration.

On the other hand, accelerated methods usually need additional parameters in comparison with slower one, which are in most cases difficult to estimate. In addition, these schemes are designed for one particular setting and cannot be used elsewhere.

In this thesis, we explore a new approach for accelerating optimization algorithms, which uses generic acceleration arguments. In numerical analysis, these tools have been developed for accelerating sequences of scalars or vectors, by building on the side another sequence with a better convergence rate. These methods can be combined with an iterative algorithm, speeding it up in most cases. In practice, extrapolation schemes are not widely used due to their lack of theoretical guarantees and their instability. We will extend these methods by regularizing them, allowing a deeper theoretical analysis and stronger convergence results, especially when applied to optimization methods.

Contents

Contributions

This thesis is divided into two main parts. Chapter 2 discusses the links between integration theory in numerical analysis and optimization methods. Chapter 3 develops a generic acceleration method for iterative algorithm, and Chapter 4 applies the results of Chapter 3 to optimization algorithms.

Chapter 1: This chapter is an introduction to optimization, integration theory and sequence acceleration in numerical analysis, which are the main topics of this thesis.

Chapter 2: This chapter introduces basics of integration theory and present standard integration methods. Then, it makes the link between these methods and optimization algorithms. These links allow an unified analysis of many optimization schemes and give new intuitions about their theoretical properties, such as improved rate of convergence. This chapter is based on the results of [66].

Chapter 3: This chapter introduces the notion of acceleration in numerical analysis, then proposes a novel algorithm for accelerating vector sequences. In comparison with previous work in this vein, this new scheme is more stable and has a proven rate of convergence. The analysis is then pushed to more complex vector sequences, thus making the acceleration method more versatile. The results of this chapter have been partially published in [64, 65, 68], and the other have been revised.

Chapter 4: This chapter applies the results of Chapter 3 to standard optimization methods, in the deterministic or stochastic setting. It also analyses the effects of acceleration to the optimization of neural networks. Then, we present some numerical experiments on different data sets using various, standard optimization algorithms to show the effectiveness of the acceleration method. The results of this chapter have been published in [64, 65, 68, 67].

Chapter 1

Introduction

In this thesis, we develop tools to analyse and improve optimization methods, especially for convex functions. We will thus recall some key results from optimization, but also from acceleration methods and integration schemes.

1.1 Convex Optimization

One of the main goals of optimization is the design of efficient algorithms for solving

$$\min_{x \in \mathcal{C}} f(x)$$

in the variable $x \in \mathcal{C} \subset \mathbb{R}^d$, where $\mathcal{C}$ is the feasible set and f is the objective function. In general, this is an impossible problem, unless we add some assumptions.

A common structural hypothesis is convexity. This makes the minimization problem tractable by generic algorithms. If an optimization problem is convex, this means that

- The set $\mathcal{C}$ is convex, i.e., $\forall x,\, y \in \mathcal{C}$, we also have $\alpha x + (1-\alpha)y \in \mathcal{C}$, $\alpha \in [0,1]$.
- The function f is convex, i.e., its epigraph $\{(x,z) : x \in \mathcal{C},\, z \geq f(x)\}$ is a convex set.

This is a sufficient condition to ensure that there exist methods, potentially slow, to solve this minimization problem. With additional structure it is possible to design more efficient algorithms with better convergence properties. For simplicity, in the rest of this thesis, we consider the case of unconstrained optimization, which means $\mathcal{C} = \mathbb{R}^d$.

1.1.1 Smoothness and Strong Convexity

Classical assumptions in optimization are smoothness (requires that f is differentiable) and (strong) convexity. We assume thorough the thesis that ∇f exists.

- **Smoothness.** The function f is smooth with constant L if, for all $x, y \in \mathbb{R}^d$,

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}\|y - x\|^2.$$

- **Strong convexity.** The function f is strongly convex with constant μ if, for all $x, y \in \mathbb{R}^d$,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2}\|y - x\|^2.$$

These definitions are valid for any norm, but we restrict here to the standard case where $\|\cdot\|$ stands for the Euclidean norm. These assumptions upper and lower bound the function by quadratics. With smoothness and strong convexity, it is possible to design efficient algorithms, with good theoretical guarantees.

1.1.2 Deterministic Optimization Algorithms

One of the oldest scheme in optimization is the so-called *steepest descent*, or gradient descent. It consists in computing the gradient at one point, then move in its opposite direction. More formally,

$$x_{i+1} = x_i - h\nabla f(x_i) \qquad \text{(Gradient Step)}$$

where h is the step size parameter. The convergence of this scheme is dependent of h and f: it is possible to show that gradient descent converges with a certain rate, function of the regularity of f, if h is well-chosen.

Proposition 1.1.1. *(Nesterov [55]) Let the sequence $\{x_i\}_{i=1\dots N}$ be generated by (4.3). Let f^* be the minimum of f. If f is smooth and convex, and if $0 < h < \frac{2}{L}$ then*

$$f(x_i) - f^* \leq \frac{2L(f(x_0) - f^*)\|x_0 - x^*\|^2}{2L\|x_0 - x^*\|^2 + i \cdot (f(x_0) - f^*)}.$$

If, in addition, f is strongly convex and $0 < h < \frac{2}{\mu + L}$, then the minimizer $x^* : f(x^*) = f^*$ is unique and

$$\|x_i - x^*\|^2 \leq \left(1 - \frac{2h\mu L}{\mu + L}\right)^i \|x_0 - x^*\|^2.$$

Despite its simplicity, the rate of convergence of gradient descent is not so bad. However, it is possible to use an improved version which uses past iterates, originally found in [50]. Its latest version is written

$$\begin{cases} x_{i+1} &= y_i - \frac{1}{L}\nabla f(y_i) \\ y_{i+1} &= (1 + \beta_i)x_i - \beta_i x_{i-1} \end{cases} \qquad \text{(Fast Gradient)}$$

where $\beta_i = \frac{i-1}{i-2}$ when the function is convex, or $\beta_i = \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}$ when we also have strong convexity. In comparison with the gradient method, this method roughly square-roots the number of

required iterations to reach ϵ-accuracy,

$$f(x) - f^* \leq \epsilon.$$

More formally, the next proposition gives the rate of convergence of the fast gradient method.

Proposition 1.1.2. *(Nesterov [55]) Let the sequence $\{x_i\}_{i=1\dots N}$ be generated by* (Fast Gradient)*. Let f^* be the minimum of f. If f is smooth and convex, and if $\beta_i = \frac{i-1}{i-2}$, then*

$$f(x_i) - f^* \leq \textit{constant} \cdot \left(\frac{1}{1+i^2}\right)\left(f(x_0) - f^* + \frac{L}{2}\|x_0 - x^*\|^2\right).$$

If, in addition, f is strongly convex and $\beta_i = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$, then

$$f(x_i) - f^* \leq \textit{constant} \cdot \left(1 - \sqrt{\frac{\mu}{L}}\right)^i \left(f(x_0) - f^* + \frac{L}{2}\|x_0 - x^*\|^2\right).$$

In the second case, the choice of β is crucial but implies the knowledge of constants μ and L. Estimating the smoothness constant is actually easy. It suffices to check if the estimate of L, says $\tilde{L}$, satisfies the descent condition

$$f\left(x - \frac{1}{\tilde{L}}\nabla f(x)\right) \leq f(x) - \frac{1}{2\tilde{L}}\|\nabla f(x)\|^2$$

when we perform a gradient step. If not, we increase $\tilde{L}$ until the condition is met. However, the strong convexity parameter is much harder in practice, and its estimation remains a challenge [25].

The algorithm (Fast Gradient) remains mysterious from an intuitive point of view. Of course, if we use more memory then we can only have a faster algorithm, but it remains unclear *why* it makes the method faster than gradient method. The proof relies on purely algebraic arguments and the key mechanism behind acceleration remains elusive, which led to various interpretations of it [13, 3, 43, 74, 41, 78, 79].

1.1.3 Stochastic Optimization

In the case of stochastic optimization, we do not have an explicit access to $\nabla f(x)$. Instead, we can only compute the noisy version

$$\nabla f_\epsilon(x) = \nabla f(x) + \varepsilon.$$

This is a typical setting in machine learning, where the objective function f is a sum of error function f_i, i.e.,

$$f(x) = \sum_{i=1}^{m} f_i(x) + R(x)$$

where $R(x)$ is a regularizer, for example, $R(x) = \|x\|^2$. Instead of the full gradient, we only compute the gradient for function f_i, where i is sampled uniformly in $\{1, \ldots, m\}$.

Stochastic optimization is typically challenging as classical algorithms are not convergent (for example, gradient descent or Nesterov's accelerated gradient). Even the averaged version of stochastic gradient descent with constant step size does not converge.

Algorithms such as SAG [63], SAGA [18], SDCA [69] and SVRG [38] accelerate convergence using a variance reduction technique akin to control variate in Monte-Carlo methods. Their rate of convergence depends on $1 - \mu/L$ and thus does not exhibit an accelerated rate on par with the deterministic setting (in $1 - \sqrt{\mu/L}$). Recently a generic acceleration algorithm called Catalyst [45], based on the proximal point methods improved this rate of convergence. On the other hand, recent papers, for example [70] (Accelerated SDCA) and [2] (Katyusha), propose algorithms with accelerated convergence rates, if the strong convexity parameter is given.

1.2 Links Between Optimization and Numerical Integration

We have seen that Nesterov's method is an efficient algorithm for minimizing a smooth, convex function f. However, its principal arguments are mostly algebraic, so it is hard to explain it with simple, intuitive arguments. On the other hand, there exist many different interpretations of gradient descent. One of the most popular way to explain it is the continuous time interpretation.

1.2.1 Gradient Flow and Euler's method

Consider the gradient descent algorithm,

$$x_{i+1} = x_i - h\nabla f(x_i).$$

We can write it so that we isolate the x's from the gradient,

$$\frac{x_{i+1} - x_i}{h} = -\nabla f(x_i).$$

If we take the limit $h \to 0$, we end up with a differential equation, called the *gradient flow*,

$$\begin{aligned} x(0) &= x_0 \\ \frac{\mathrm{d}x}{\mathrm{d}t} \triangleq \dot{x} &= -\nabla f(x(t)). \end{aligned} \qquad \text{(Gradient Flow)}$$

From a geometrical point of view, the gradient flow equation tells us that the trajectory of $x(t)$ follows always the steepest descent, similarly to the trajectory of a water drop in a valley, and this corresponds quite accurately to the discrete trajectory of x_i.

In fact, the discrete gradient descent corresponds exactly to the integration of (Gradient Flow) when using the well-known Euler's method. When we face an ordinary differential equation of the type

$$\frac{\mathrm{d}x}{\mathrm{d}t} = g(x(t)), \tag{1.1}$$

Euler's method estimates $\frac{\mathrm{d}x}{\mathrm{d}t}$ using the forward difference

$$\frac{\mathrm{d}x}{\mathrm{d}t} \approx \frac{x_{i+1} - x_i}{t_{i+1} - t_i}, \quad x(t) \approx x(t_i).$$

If we assume $t_{i+1} - t_i = h$ constant, we end up with

$$x_{i+1} = x_i + hg(x_i).$$

Here, we see a perfect match between Euler's method and gradient scheme when $g = -\nabla f$. However, when moving to more complex algorithm such as Nesterov's fast gradient, things get more complicated.

1.2.2 Interpretation of Fast Gradient Algorithm

The main issue with the explanation of fast gradient descent using continuous time interpretation is the following: we can either refine the differential equation, or the integration method.

For example, on the basis of the work done by Su et al. [74], [78] proposed to analyse a complex differential equation, based on the *Bregman Lagrangian* function, written (in the Euclidean case)

$$\mathcal{L}(x, v, t) = e^{\alpha_t + \gamma_t} \left(\|e^{-\alpha_t} v\|_2^2 - e^{-\beta_t} f(x) \right),$$

where α_t, β_t and γ_t follows some "ideal scaling" conditions. Then, the differential equation is written

$$\frac{\mathrm{d}}{\mathrm{d}t} \left(x(t) + e^{-\alpha_t} \frac{\mathrm{d}x(t)}{\mathrm{d}t} \right) = -e^{\alpha_t + \beta_t} \nabla f(x(t)).$$

In their work, the authors proved an accelerated rate in continuous time for this differential equation. Then they propose a smart, but unusual, discretization based on Euler's scheme which preserves the rate of convergence in the discrete case.

In our work [66], we propose to keep the simple and intuitive differential equation (Gradient Flow). Instead, we will use a different integration rule to approximate the curve $x(t)$. There exists two big families of integration methods, named *Runge-Kutta* and *Linear Multistep* methods. They all are well-studied, and used in many applications. For example in the MATLAB software the function ODE45 is a specific instance of *Runge-Kutta* method.

In this thesis, we will introduce the theory of integration methods and make the parallel with existing optimization scheme, and show that acceleration can be explained by the usage of larger step size. We focus on the family of (explicit) *Linear Multistep* methods for integrating (1.1), written

$$\sum_{i=1}^{s} \rho_i x_i = h \sum_{i=1}^{s-1} \sigma_i g(x_i)$$

for some coefficients ρ, σ and step-size h (later, ρ and σ will represent polynomials). The previous method is explicit because we can compute x_s using the past iterates only, since

$$x_s = \frac{1}{\rho_s} \left(\sum_{i=1}^{s-1} \rho_i x_i - h \sum_{i=1}^{s-1} \sigma_i g(x_i) \right).$$

It happens there is a perfect match between many optimization algorithms and this family of methods, when g is actually $-\nabla f$. For example, the Nesterov's method can be written

$$x_{i+2} = \underbrace{\frac{1}{\beta}}_{=\frac{1}{\rho_s}} \left(\underbrace{(1+\beta)x_{i+1} - x_{i+2}}_{=\sum_{i=1}^{s-1} \rho_i x_i} + \underbrace{\frac{1}{L}\left(-\beta\left(-\nabla f(x_{i+1})\right) + (1+\beta)\left(-\nabla f(x_{i+1})\right)\right)}_{=\frac{1}{h}\sum_{i=1}^{s-1} \sigma_i g(x_i)} \right)$$

where $\beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$. We clearly see that Nesterov's method can also be an instance of a linear multi-step method. In addition, if we assume that the "Nesterov integration scheme" is stable and consistent (two important conditions that almost all integration methods should meet, they basically ask the method to integrate perfectly first order polynomials), we can retrieve its step size h. Surprisingly, for smooth and strongly convex problem, the step size is $\sqrt{\frac{L}{\mu}}$ larger than the step size of gradient descent, which means the method integrates the differential equation $\sqrt{\frac{L}{\mu}}$ times *faster*. Again, this corresponds to the theoretical speedup of Nesterov's method over the gradient scheme.

1.3 Acceleration Methods and Optimization Algorithms

We have seen that accelerated methods in optimization exist, and present an improved rate of convergence. However, their main weakness is their lack of adaptivity to the problem constants. For example, in the case of Nesterov's method, we have seen it is possible to estimate the Lipchitz constant while the algorithm is running, but not the strong convexity parameter. If we have acess to a good estimate of the strong convexity constant, this is not a problem. However, if it is not the case, we have to choose between

- the strong-convex variant of Nesterov's scheme ($\beta_i = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$) with a bad estimate of μ, which leads to a bad convergence rate or

- the convex variant of Nesterov's scheme ($\beta_i = \frac{i-1}{i-2}$), which does not have a linear rate of convergence.

On the other hand, there exist generic acceleration methods such as Catalyst [45] or LBFGS [11]. However, the numerical performance of Catalyst is not as good as the Fast gradient method, and LBFGS lacks theoretical guarantees.

1.3.1 Anderson Acceleration

In this thesis, we propose a generic acceleration method based on Anderson Acceleration [4]. This method is capable to improve the rate of an iterative algorithm of the form

$$x_{i+1} = g(x_i)$$

where the Jacobian of g is symmetric. For example, this corresponds to the gradient descent where g is

$$g(x) = x - h\nabla f(x).$$

The main idea of Anderson acceleration is to combine the previous iterates x_i with coefficient c_i to form an *extrapolation* x_{extr}. This extrapolation aims to approximate the solution x^*. In other words, Anderson acceleration computes c_i such that

$$x_{\text{extr}} = \sum_{i=0}^{N} c_i x_i \approx x^*.$$

As opposed to Nesterov's method, where the coefficient β_i are known in advance, the coefficients c_i are computed using only the iterates x_i. However, the result has major stability issues (a small noise in the iterates x_i produces large perturbation on the extrapolation x_{extr}), and thus Anderson acceleration is not suitable in most cases.

1.3.2 Regularized Nonlinear Acceleration

In Chapter 3, we introduce the *Regularized Nonlinear Acceleration* technique, which is basically a regularized version of Anderson acceleration. This makes the algorithm more stable, with proven convergence bound (Theorem 3.3.9). In addition regularization makes the acceleration scheme robust to any kind of noise, so we can derive a similar bound for stochastic algorithms (Proposition 3.3.11). Finally, we identify a larger class of algorithm, written

$$\begin{aligned} x_{i+1} &= g(y_i), \\ y_{i+1} &= \sum_{j=1}^{i+1} \alpha_j^{(i)} x_j + \sum_{j=0}^{i} \beta_j^{(i)} y_j. \end{aligned}$$

where the coefficients α and β are arbitrary. This class contains many different optimization schemes, such as gradient method but more importantly Nesterov's algorithm. We then propose a novel method which "accelerates" this family of methods, while keeping their theoretical properties (Proposition 3.5.3 and Algorithm 5).

In Chapter 4, we apply the results of Chapter 3 to optimization methods for deterministic and stochastic algorithms, and also on neural networks. In particular, we show this acceleration technique improves the rate of convergence of many algorithms, and usually outperforms optimal methods such as the fast gradient in the deterministic case, or Katyusha [2] for the stochastic setting.

Chapter 2

Integration Methods and Optimization Algorithms

Abstract

We show that accelerated optimization methods can be seen as particular instances of multi-step integration schemes from numerical analysis, applied to the gradient flow equation. In comparison with recent advances in this vein, the differential equation considered here is the basic gradient flow and we show that multi-step schemes allow integration of this differential equation using larger step sizes, thus intuitively explaining acceleration results.

Main reference: The contributions of this chapter are published in the NIPS 2017 conference [66].

Collaboration with Vincent Roulet: This work was done in collaboration with Vincent Roulet, and we both equally contributed to this paper.

2.1 Introduction

The gradient descent algorithm used to minimize a function f has a well-known simple numerical interpretation as the integration of the gradient flow equation, written

$$\begin{aligned} x(0) &= x_0 \\ \dot{x}(t) &= -\nabla f(x(t)), \end{aligned} \tag{Gradient Flow}$$

using Euler's method. This appears to be a somewhat unique connection between optimization and numerical methods, since these two fields have inherently different goals. On one hand,

numerical methods aim to get a precise discrete approximation of the solution $x(t)$ on a finite time interval. More sophisticated methods than Euler's were developed to get better consistency with the continuous time solution but still focus on a finite time horizon [see for example 75]. On the other hand, optimization algorithms seek to find the minimizer of a function, which corresponds to the infinite time horizon of the gradient flow equation. Structural assumptions on f led to more sophisticated algorithms than the gradient method, such as the mirror gradient method [see for example 8, 6], proximal gradient method [54] or a combination thereof [21, 56]. Among them Nesterov's accelerated gradient algorithm [52] is proven to be optimal on the class of smooth convex or strongly convex functions. This last method was designed with the lower complexity bounds in mind, but the proof relies on purely algebraic arguments and the key mechanism behind acceleration remains elusive, which led to various interpretations of it [13, 3, 43].

A recent stream of papers used differential equations to model the acceleration behavior and offer a better interpretation of Nesterov's algorithm [74, 41, 78, 79]. However, the differential equation is often quite complex, being reverse-engineered from Nesterov's method itself, thus losing the intuition. Moreover, integration methods for these differential equations are often ignored or are not derived from standard numerical integration schemes, because the convergence proof of the algorithm does not require the continuous time interpretation.

Here, we take another approach. Rather than using a complicated differential equation, we use advanced multi-step methods to discretize the basic gradient flow equation in (Gradient Flow). These lesser known methods, developed decades ago by numerical analysts, directly correspond to various well-known optimization algorithms. In particular, Nesterov's method can be seen as a stable (a small noise in the initial conditions produces bounded perturbations) and consistent (the limit of the numerical approximation when $h \to 0$ is the differential equation) gradient flow discretization scheme that allows bigger step sizes in integration, leading to faster convergence.

The chapter is organized as follows. In Section 2.2 we present our setting and recall classical results on differential equations. We then review definitions and theoretical properties of integration methods called linear multi-step methods in Section 2.3. Linear one-step and two-step methods are detailed in Section 2.4 and linked to optimization algorithms in Section 2.5 (strongly convex case) and Section 2.6 (convex case). Finally, we propose several extensions for (Gradient Flow), which can be used to explain proximal methods (Section 2.7), non-euclidean method (Section 2.8) or a combination thereof (Section 2.9).

2.2 Gradient flow

We seek to minimize a L-smooth and μ-strongly convex function f defined on $\mathbb{R}^d$ and look in that purpose at the discretization the gradient flow equation (Gradient Flow), given by the following

ordinary differential equation (ODE)

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}x(t) \triangleq \dot{x}(t) &= g(x(t)) \\ x(0) &= x_0, \end{aligned} \tag{ODE}$$

where g comes from a potential $-f$, meaning $g = -\nabla f$. Smoothness of f means Lipschitz continuity of g, i.e.

$$\|g(x) - g(y)\| \leq L\|x - y\|, \quad \text{for every } x, y \in \mathbb{R}^d,$$

where $\|.\|$ is the Euclidean norm. This property ensures the existence and uniqueness of the solution of (ODE) (see [75, Theorem 12.1]). Strong convexity of f means strong monotonicity of $-g$, i.e.,

$$\mu\|x - y\|^2 \leq -\langle x - y, g(x) - g(y)\rangle, \quad \text{for every } x, y \in \mathbb{R}^d,$$

and ensures that (ODE) has a unique point x^* such that $g(x^*) = 0$, called the equilibrium. This is the minimizer of f and the limit point of the solution, i.e., $x(\infty) = x^*$. In the numerical analysis literature, strong monotonicity of $-g$ is referred to as one-sided Lipschitz-continuity of g. Whatever its name, this property essentially contracts solutions of (ODE) with different initial points as showed in the following proposition, where μ controls the rate of contraction.

Proposition 2.2.1. *Assume that $x_1(t), x_2(t)$ are solutions of ODE with different initial conditions, where $-g$ is strongly monotone. Then, $x_1(t)$ and $x_2(t)$ get closer as t increases, precisely,*

$$\|x_1(t) - x_2(t)\|^2 \leq e^{-2\mu t}\|x_1(0) - x_2(0)\|^2.$$

Proof. Let $\mathcal{L}(t) = \|x_1(t) - x_2(t)\|^2$ If we derive $\mathcal{L}(t)$ over time,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\mathcal{L}(t) &= 2\langle\, x_1(t) - x_2(t), \dot{x}_1(t) - \dot{x}_2(t)\rangle \\ &= 2\langle\, x_1(t) - x_2(t), g(x_1(t)) - g(x_2(t))\rangle \\ &\leq -2\mu\mathcal{L}(t). \end{aligned}$$

We thus have $\mathcal{L}(t) \leq e^{-2\mu t}\mathcal{L}(0)$, which is the desired result. ■

Strong convexity allows us to control the convergence rate of the potential f and the solution $x(t)$ as recalled in the following proposition.

Proposition 2.2.2. *Let f be a L-smooth and μ-strongly convex function and $x_0 \in \mathbf{dom}(f)$. Writing x^* the minimizer of f, the solution $x(t)$ of* (Gradient Flow) *satisfies*

$$f(x(t)) - f(x^*) \leq (f(x_0) - f(x^*))e^{-2\mu t} \tag{2.1}$$

$$\|x(t) - x^*\| \leq \|x_0 - x^*\|e^{-\mu t}. \tag{2.2}$$

Proof. Indeed, if we derive the left-hand-side of (2.1),

$$\frac{\mathrm{d}}{\mathrm{d}t}[f(x(t)) - f(x^*)] = \langle \nabla f(x(t)), \dot{x}(t) \rangle = -\|\nabla f(x(t))\|^2.$$

Using that f is strongly convex, we have (see [55])

$$f(x) - f(x^*) \leq \frac{1}{2\mu}\|\nabla f(x)\|^2,$$

and therefore

$$\frac{\mathrm{d}}{\mathrm{d}t}[f(x(t)) - f(x^*)] \leq -2\mu[f(x(t)) - f(x^*)].$$

Solving this differential equation leads to the desired result. We can apply a similar technique for the proof of (2.2), using that

$$\mu\|x - y\|^2 \leq \langle \nabla f(x) - \nabla f(y), x - y \rangle,$$

for strongly convex functions (see again [55]). ■

We now focus on numerical methods to integrate (ODE).

2.3 Numerical integration of differential equations

In general, we do not have access to an explicit solution $x(t)$ of (ODE). We thus use integration algorithms to approximate the curve $(t, x(t))$ by a grid $(t_k, x_k) \approx (t_k, x(t_k))$ on a finite interval $[0, t_{\max}]$. For simplicity here, we assume the step size $h_k = t_k - t_{k-1}$ constant, i.e., $h_k = h$ and $t_k = kh$. The goal is then to minimize the approximation error $\|x_k - x(t_k)\|$ for $k \in [0, t_{\max}/h]$. We first introduce Euler's explicit method to illustrate this on a basic example.

2.3.1 Euler's explicit method

Euler's explicit method is one of the oldest and simplest schemes for integrating the curve $x(t)$. The idea stems from the Taylor expansion of $x(t)$ which reads

$$x(t + h) = x(t) + h\dot{x}(t) + O(h^2).$$

When $t = kh$, Euler's explicit method approximates $x(t + h)$ by x_{k+1} by neglecting the second order term,

$$x_{k+1} = x_k + hg(x_k).$$

In optimization terms, we recognize the gradient descent algorithm used to minimize f. Approximation errors in an integration method accumulate with iterations, and as Euler's method uses

only the last point to compute the next one, it has only limited control over the accumulated error.

2.3.2 Linear multi-step methods

Multi-step methods use a combination of several past iterates to improve convergence. Throughout the paper, we focus on *linear* s-step methods whose recurrence can be written

$$x_{k+s} = -\sum_{i=0}^{s-1} \rho_i x_{k+i} + h \sum_{i=0}^{s} \sigma_i g(x_{k+i}), \quad \text{for } k \geq 0, \tag{2.3}$$

where $\rho_i, \sigma_i \in \mathbb{R}$ are coefficients of polynomials ρ, σ and h is again the step size. Each new point x_{k+s} is a function of the information given by the s previous points. If $\sigma_s = 0$, each new point is given explicitly by the s previous points and the method is then called **explicit**. Otherwise each new point requires solving an implicit equation and the method is then called **implicit**.

To simplify notation we use the shift operator E, which maps $Ex_k \to x_{k+1}$. Moreover, if we write $g_k = g(x_k)$, then the shift operator also maps $Eg_k \to g_{k+1}$. Recall that a univariate polynomial is called monic if its leading coefficient is equal to 1. We now give the following concise definition of s-step linear methods.

Definition 2.3.1. *A **linear s-step method** integrates an* (ODE) *defined by g, x_0, using a step size h and $x_1, \ldots, x_{s-1}$ starting points by generating a sequence x_k that satisfies*

$$\rho(E)x_k = h\sigma(E)g_k, \quad \textit{for every } k \geq 0,$$

where ρ is a monic polynomial (i.e. its leading coefficient is equal to 1) of degree s with coefficients ρ_i, and σ a polynomial of degree s with coefficients σ_i.

A linear $s-$step method is uniquely defined by the polynomials (ρ, σ). The sequence generated by the method then depends on the starting points and the step size. Each linear multi step method has a twin sister, called *one-leg* method, which generates the sequence $\tilde{x}_k$ following

$$\rho(E)\tilde{x}_k = \sigma(1)hg\left(\frac{\sigma(E)}{\sigma(1)}\tilde{x}_k\right).$$

It is possible to show a bijection between x_k and $\tilde{x}_k$ [17]. We quickly mention one-leg methods in Section 2.7 but we will not go into more details here. We now recall a few results describing the performance of multi-step methods.

2.3.3 Stability

Stability is a key concept for integration methods. First of all, consider two curves $x(t)$ and $y(t)$, both solutions of an (ODE) defined by g, but starting from different points $x(0)$ and $y(0)$. If the

function g is Lipchitz-continuous (but not especially monotone), we have show in Proposition 2.2.1 that the distance between $x(t)$ and $y(t)$ is bounded on a finite interval, i.e.

$$\|x(t) - y(t)\| \leq C\|x(0) - y(0)\| \qquad \forall t \in [0, t_{\max}],$$

where C may depend exponentially on $t_{\max}$. We would like to have a similar behaviour for our sequences x_k and y_k, approximating $x(t_k)$ and $y(t_k)$, i.e.

$$\|x_k - y_k\| \approx \|x(t_k) - y(t_k)\| \leq C\|x(0) - y(0)\| \qquad \forall k \in [0, t_{\max}/h], \tag{2.4}$$

when $h \to 0$, so $k \to \infty$. Two issues quickly arise, namely:

- For a linear s-step method, we need s starting values $x_0, ..., x_{s-1}$. Condition (2.4) will therefore depend on all these starting values and not only x_0.

- Any discretization scheme introduces at each step an approximation error, called local error, which is accumulated over time. We denote this error by $\epsilon_{\text{loc}}(x_{k+s})$ and define it as

  $$\epsilon_{\text{loc}}(x_{k+s}) \triangleq x_{k+s} - x(t_{k+s})$$

  if x_{k+s} is computed using $x_k = x(t_k), ..., x_{k+s-1} = x(t_{k+s-1})$, where $x(t)$ is the solution of the differential equation.

In other words, the difference between x_k and y_k can be described as follows

$$\|x_k - y_k\| \ \leq \ \text{Error in the initial condition} + \text{Accumulation of local errors.}$$

We now write a complete definition of stability, inspired from Definition 6.3.1 from Gautschi [28].

Definition 2.3.2. *A linear multi-step method is stable if, for two sequences x_k, y_k generated by (ρ, σ) using any sufficiently small step size $h > 0$, from the starting values $x_0, ..., x_{s-1}$, and $y_0, ..., y_{s-1}$, we have*

$$\|x_k - y_k\| \leq C \left(\max_{i \in \{0,...,s-1\}} \|x_i - y_i\| + \sum_{i=1}^{t_{\max}/h} \|\epsilon_{\text{loc}}(x_{i+s}) - \epsilon_{\text{loc}}(y_{i+s})\| \right), \tag{2.5}$$

for any $k \in [0, t_{\max}/h]$. Here, the constant C may depend on $t_{\max}$ but is independent of h.

When h tends to zero, we may recover equation (2.4) only if the accumulated local error tends also to zero. We thus need

$$\lim_{h \to 0} \ \frac{1}{h} \ \|\epsilon_{\text{loc}}(x_{i+s}) - \epsilon_{\text{loc}}(y_{i+s})\| = 0 \quad \forall i \in [0, t_{\max}/h].$$

This condition is called *consistency*. Once this condition satisfied, we still need to ensure

$$\|x_k - y_k\| \leq C \max_{i \in \{0,\dots,s-1\}} \|x_i - y_i\|, \tag{2.6}$$

and this condition is called *zero-stability* (because, surprisingly, it suffices to check this condition for the special case $g(x) = 0$).

Truncation error and consistency

The truncation error of a linear multi-step method is a measure of the local error $\epsilon_{\text{loc}}(x_k)$ made by the method, normalized by h. More precisely it is defined using the difference between the step performed by the algorithm and the step which reaches exactly $x(t_{k+s})$, with

$$T(h) \triangleq \frac{x(t_{k+s}) - x_{k+s}}{h} \qquad \text{assuming } x_{k+i} = x(t_{k+i}),\ i = 0, \dots, s-1. \tag{2.7}$$

This definition does not depend on k but on the recurrence of the linear s-step method and on the (ODE) defined by g and x_0. We can use this truncation error to define consistency.

Definition 2.3.3. *An integration method for an* (ODE) *defined by g, x_0 is consistent if and only if, for any initial condition x_0,*

$$\lim_{h \to 0} \|T(h)\| = 0.$$

The following proposition shows there exist simple conditions to check consistency, which rely on comparing a Taylor expansion of the solution with the coefficients of the method.

Proposition 2.3.4. *A linear multi-step method defined by polynomials (ρ, σ) is consistent if and only if*

$$\rho(1) = 0 \qquad \text{and} \qquad \rho'(1) = \sigma(1). \tag{2.8}$$

Proof. Assume $t_k = kh$. If we expand $g(x(t_k))$ we have,

$$g(x(t_k)) = g(x_0) + O(h).$$

If we do the same thing with $x(t_k)$, we have given (2.3),

$$x(t_k) = x_0 + kh\dot{x}(t_0) + O(h^2) = x_0 + khg(x_0) + O(h^2).$$

If we plug these results in the linear multi-step method,

$$\begin{aligned} T(h) &= \frac{1}{h}\left(x(t_{k+s}) + \sum_{i=0}^{s-1} \rho_i x(t_{k+i}) - h \sum_{i=0}^{s} \sigma_i g(x(t_{k+i})) \right) \\ &= \frac{1}{h}\rho(1)x_0 + (\rho'(1) - \sigma(1))g(x_0) + O(h). \end{aligned}$$

The limit is equal to zero if and only if we satisfy (2.8). ■

Consistency is crucial for integration methods, we give some intuition about how important are conditions defined in (2.8).

First condition, $\rho(1) = 0$. If the condition is not satisfied, then the method exhibits an artificial gain or damping. Assume we start at some equilibrium x^* of ODE (i.e. $\nabla f(x^*) = 0$), so $x_i = x^*$ for the first $s-1$ steps. The next iterate becomes

$$x_s = -\sum_{i=0}^{s-1} \rho_i x^* + h\sigma(E) \underbrace{g(x^*)}_{=0},$$

and if $1 + \sum_{i=0}^{s} \rho_i = \rho(1) \neq 0$, we have that the next iterate x_s is different from x^*.

Second condition, $\rho'(1) = \sigma(1)$. If this relation is not satisfied, we actually are integrating another equation than (ODE) as shown in the following reasoning. Assume the first condition satisfied, 1 is a root of ρ. Consider then the factorization

$$\rho(z) = (z-1)\tilde{\rho}(z)$$

where $\tilde{\rho}$ is a polynomial of degree $s-1$, and $\rho'(1) = \tilde{\rho}(1)$. The linear multi-step method becomes

$$\tilde{\rho}(E)(y_{k+1} - y_k) = h\sigma(E)g(y_k).$$

If we sum up the above equation from the initial point, we get

$$\tilde{\rho}(E)(y_k - y_0) = \sigma(E)G_k,$$

where $G_k = \sum_{i=0}^{k} hg(y_i)$. If h goes to zero, our iterates y_k converge to a continuous curve $y(t)$, and $G_k \to \int_0^t g(y(\tau))\,\mathrm{d}\tau$,

$$\sum_{i=0}^{s} \tilde{\rho}_i(y(t) - x(0)) = \sum_{i=0}^{s-1} \sigma_i \int_0^t g(y(\tau))\,\mathrm{d}\tau.$$

If we take the derivative over time, we get

$$\tilde{\rho}(1)\dot{y}(t) = \sigma(1)g(y(t)) \quad \Leftrightarrow \quad \rho'(1)\dot{y}(t) = \sigma(1)g(y(t)).$$

which is different from the ODE we wanted to discretize, unless $\rho'(1) = \sigma(1)$.

Zero-stability and root condition

To get stability, assuming consistency holds as above, we also need to satisfy the zero-stability condition (2.6), which characterizes the sensitivity of a method to initial conditions. Actually, the name comes from an interesting fact: analyzing the special case where $g = 0$ is completely equivalent to the general case, as stated in the *root condition theorem*.

Theorem 2.3.5 (Root condition)**.** *Consider a linear multi-step method (ρ, σ). The method is zero-stable if and only if all roots of ρ are in the unit disk, and the roots on the unit circle are simple.*

The proof of this theorem is technical and can be found as Theorem 6.3.4 of Gautschi [28]. The analysis becomes simplified, and reduces to standard linear algebra results because we only need to look at the solution of the homogeneous difference equation $\rho(E)x_k = 0$.

2.3.4 Convergence of the global error and Dahlquist's equivalence theorem

Numerical analysis focuses on integrating an ODE on a finite interval of time $[0, t_{\max}]$. It studies the behavior of the global error defined by the difference between the scheme and the solution of the ODE, i.e., $x(t_k) - x_k$, as a function of the step size h. If the global error converges to 0 with the step size, the method is guaranteed to approximate correctly the ODE on the time interval, for h small enough. The faster it converges with h, the less points we need to guarantee a global error within a given accuracy.

We now state *Dahlquist's equivalence theorem*, which shows that the global error converges to zero when h does if the method is *stable*, i.e., when the method is *consistent* and *zero-stable*. This naturally needs the additional assumption that the starting values $x_0, \ldots, x_{s-1}$ are computed such that they converge to the solution $(x(0), \ldots, x(t_{s-1}))$.

Theorem 2.3.6 (Dahlquist's equivalence theorem)**.** *Given an* (ODE) *defined by g and x_0 and a consistent linear multi-step method (ρ, σ), whose starting values are computed such that $\lim_{h\to 0} x_i = x(t_i)$ for any $i \in \{0, \ldots, s-1\}$, zero-stability is necessary and sufficient for being convergent, i.e., $x(t_k) - x_k$ tends to zero for any k when the step size h tends to zero.*

Again, the proof of the theorem can be obtained from Gautschi [28]. Notice that this theorem is fundamental in numerical analysis. For example, it says that if a *consistent* method is not zero-stable, then the global error may be arbitrary large when the step size goes to zero, *even if the local error decreases*. In fact, if zero-stability is not satisfied, there exists a sequence generated by the linear multi-step method which grows with arbitrarily large factors.

2.3.5 Region of absolute stability

Stability and global error are ensured on finite time intervals, however solving optimization problems requires us to look at the infinite time horizon. We thus need more refined results and

start by finding conditions ensuring that the numerical solution does not diverge when the time interval increases, i.e. that the numerical solution is stable with a constant C which *does not depend of* $t_{\max}$. Formally, for a fixed step-size h, we want to ensure

$$\|x_k\| \leq C \max_{i\in\{0,\dots,s-1\}} \|x_i\| \quad \text{for all } k \in [0, t_{\max}/h] \text{ and } t_{\max} > 0. \tag{2.9}$$

This is not possible without assumptions on the function g as in the general case the solution $x(t)$ itself may diverge. We begin with the simple scalar linear case which, given $\lambda > 0$, reads

$$\begin{aligned} \dot{x}(t) &= -\lambda x(t) \\ x(0) &= x_0. \end{aligned} \tag{Scalar Linear ODE}$$

The recurrence of a linear multi-step methods with parameters (ρ, σ) applied to (Scalar Linear ODE) then reads

$$\rho(E)x_k = -\lambda h\sigma(E)x_k \quad \Leftrightarrow \quad [\rho + \lambda h\sigma](E)x_k = 0,$$

where we recognize an homogeneous recurrence equation. Condition (2.9) is then controlled by the step size h and the constant λ, ensuring that this homogeneous recurrent equation produces bounded solutions. This leads us to the definition of the region of absolute stability.

Definition 2.3.7. *The region of absolute stability of a linear multi-step method defined by* (ρ, σ) *is the set of values* λh *such that the characteristic polynomial*

$$\pi_{\lambda h} \triangleq \rho + \lambda h\sigma \tag{2.10}$$

of the homogeneous recurrent equation $\pi_{\lambda h}(E)x_k = 0$ *produces bounded solutions.*

Standard linear algebra links this condition to the roots of the characteristic polynomial as recalled in the next proposition (see Lemma 12.1 of Süli and Mayers [75]).

Proposition 2.3.8. *Let* π *be a polynomial and write* x_k *a solution of the homogeneous recurrent equation* $\pi(E)x_k = 0$ *with arbitrary initial values. If all roots of* π *are inside the unit disk and the ones on the unit circle have a multiplicity exactly equal to one, then* $\|x_k\| \leq \infty$.

Absolute stability of a linear multi-step method determines its ability to integrate a linear ODE defined by

$$\begin{aligned} \dot{x}(t) &= -Ax(t) \\ x(0) &= x_0, \end{aligned} \tag{Linear ODE}$$

where A is a positive definite matrix whose eigenvalues belong to $[\mu, L]$ for $0 < \mu \leq L$. In this case the step size h must indeed be chosen such that for any $\lambda \in [\mu, L]$, λh belongs to the region of absolute stability of the method. This (Linear ODE) is a special instance of (Gradient Flow)

where f is a quadratic function. Therefore absolute stability gives necessary (but not sufficient) condition to integrate (Gradient Flow) of L-smooth μ-strongly convex functions.

2.3.6 Convergence analysis in the linear case

By construction, absolute stability also gives us hints on the convergence of x_k to the equilibrium. More precisiely, it allows us to control the rate of convergence of x_k, approximating the solution $x(t)$ of (Linear ODE) (see Lemma 12.1 of Süli and Mayers [75]).

Proposition 2.3.9. *Given a* (Linear ODE) *defined by x_0 and a positive definite matrix A whose eigenvalues belong to $[\mu, L]$ for $0 < \mu \leq L$, for a fixed step size h and a linear multi-step method defined by (ρ, σ), let $r_{\max}$ be defined as*

$$r_{\max} = \max_{\lambda \in [\mu, L]} \max_{r \in \text{roots}(\pi_{\lambda h}(z))} |r|,$$

where $\pi_{\lambda h}$ is defined in (2.10)*. If $r_{\max} < 1$ and its multiplicity is equal to m, then the speed of convergence of the sequence x_k produced by the linear multi-step method to the equilibrium x^* of the differential equation is given by*

$$\|x_k - x^*\| = O(k^{m-1} r_{\max}^k). \tag{2.11}$$

We can now use these properties to analyze and design multi-step methods.

2.4 Analysis and design of multi-step methods

As shown before, we want to integrate (Gradient Flow) and Proposition 2.2.2 gives us a rate of convergence in the continuous case. If the method tracks $x(t)$ with sufficient accuracy, then the rate of the method will be close to the rate of convergence of $x(kh)$. So, *larger values of h yield faster convergence of $x(t)$ to the equilibrium x^**. However h cannot be too large, as the method may be too inaccurate and/or unstable as h increases. *Convergence rates of optimization algorithms are thus controlled by our ability to discretize the gradient flow equation using large step sizes.* We recall the different conditions that proper linear multi-step methods should follow.

- *Monic polynomial (Section 2.3.2).* This is a convention, otherwise dividing both sides of the difference equation of the multi-step method by ρ_s does not change the method.
- *Explicit method (Section 2.3.2).* We assume that the scheme is explicit in order to avoid solving a non-linear system at each step (Section 2.7 shows that implicit methods are linked to proximal methods).
- *Consistency (Section 2.3.3).* If the method is not consistent, then the local error does not converge when the step size goes to zero.

- *Zero-stability (Section 2.3.3).* Zero-stability ensures convergence of the global error (Section 2.3.4) when the method is also consistent.

- *Region of absolute stability (Section 2.3.5).* If λh is not inside the region of absolute stability for any $\lambda \in [\mu, L]$, then the method is divergent when $t_{\max}$ increases.

Using the remaining degrees of freedom, we will tune the algorithm to have the best rate of convergence on (Linear ODE), which corresponds to the optimization of a quadratic function. Indeed, as showed in Proposition 2.3.9, the largest root of $\pi_{\lambda h}$ gives us the rate of convergence on quadratic functions (when $\lambda \in [\mu, L]$). Since smooth and strongly convex functions are close to be quadratic (they are in fact sandwiched between two quadratics), this will also give us a good idea of the rate of convergence on these functions.

We do not derive a proof of convergence of the sequence for a general smooth and (strongly) convex function (in fact, it is already proved by [55] or using Lyapunov techniques by Wilson et al. [79]). But our results provide intuition on *why* accelerated methods converge faster.

2.4.1 Analysis and design of explicit Euler's method ($s = 1$)

In Section 2.3.1 we introduced Euler's method. In fact, we can view it as an explicit linear "multi-step" method with $s = 1$ defined by the polynomials

$$\rho(z) = -1 + z, \quad \sigma(z) = 1.$$

We can check easily that it is consistent (using Proposition 2.3.4) and zero-stable since ρ has only one root which lies on the unit circle (Theorems 2.3.5 and 2.3.6). We need to determine the region of absolute stability in order to have an idea about the maximum value that $h > 0$ can take before the method becomes unstable. Assume we want to integrate any μ-strongly convex and L-smooth function f, with $0 < \mu < L$ with any starting value x_0. Then, we need to find the set of values of h such that the roots of the polynomial

$$\pi_{\lambda h}(z) = [\rho + \lambda h \sigma](z) = -1 + \lambda h + z, \quad \lambda \in [\mu, L]$$

are small. The unique root is $1 - \lambda h$ and we need to solve the following minimax problem

$$\min_h \max_{\lambda \in [\mu, L]} |1 - \lambda h| ,$$

in the variable $h > 0$. The solution of this optimization problem is $h^* = \frac{2}{L+\mu}$, its optimal value is $(L - \mu)/(L + \mu)$ and its rate of convergence is then

$$\|x_k - x^*\| = O\left(\left(\frac{1 - \mu/L}{1 + \mu/L}\right)^k\right).$$

We recover the optimal step size and the rate of convergence of the gradient method for a general smooth and strongly convex function [55].

2.4.2 Analysis of two-step methods ($s = 2$)

We will now analyze two-step methods. First we write the conditions of a good linear multi step method, introduced at the beginning of this section, into constraints on the coefficients.

$$\begin{aligned}
\rho_2 &= 1 && \text{(Monic polynomial)}\\
\sigma_2 &= 0 && \text{(Explicit method)}\\
\rho_0 + \rho_1 + \rho_2 &= 0 && \text{(Consistency)}\\
\sigma_0 + \sigma_1 + \sigma_2 &= \rho_1 + 2\rho_2 && \text{(Consistency)}\\
|\text{Roots}(\rho)| &\leq 1 && \text{(Zero-stability).}
\end{aligned}$$

If we use all equalities, we finally have three linear constraints, defined by

$$\mathcal{L} = \Big\{\rho_0,\, \rho_1,\, \sigma_1 : \rho_1 = -(1+\rho_0); \quad \sigma_1 = 1 - \rho_0 - \sigma_0; \quad |\rho_0| < 1\Big\}. \tag{2.12}$$

We will now try to find some condition on the remaining parameters in order to have a stable method. At first, let us analyze a condition on the roots of second order equations. Absolute stability requires that all roots of the polynomial $\pi_{\lambda h}$ are inside the unit circle. The following proposition gives us the values of the roots of $\pi_{\lambda h}$ as a function of the parameters ρ_i and σ_i.

Proposition 2.4.1. *Given constants $0 < \mu \leq L$, a step size $h > 0$ and a linear two-step method defined by (ρ, σ), under the conditions*

$$\begin{aligned}
(\rho_1 + \mu h\sigma_1)^2 &\leq 4(\rho_0 + \mu h\sigma_0),\\
(\rho_1 + Lh\sigma_1)^2 &\leq 4(\rho_0 + Lh\sigma_0),
\end{aligned}$$

the roots $r_\pm(\lambda)$ of $\pi_{\lambda h}$, defined in (2.10), are complex for any $\lambda \in [\mu, L]$. Moreover, the largest modulus root is equal to

$$\max_{\lambda\in[\mu,L]} |r_\pm(\lambda)|^2 = \max\left\{\rho_0 + \mu h\sigma_0,\ \rho_0 + Lh\sigma_0\right\}. \tag{2.13}$$

Proof. We begin by analyzing the roots $r_\pm$ of the generic polynomial

$$z^2 + bz + c,$$

where b and c are real numbers, corresponding to the coefficients of $\pi_{\lambda h}$, i.e. $b = \rho_1 + \lambda h\sigma_1$ and

$c = \rho_0 + \lambda h \sigma_0$. For a fixed λ roots are complex if and only if

$$b^2 \leq 4c \quad \Leftrightarrow \quad (\rho_1 + \lambda h \sigma_1)^2 - 4(\rho_0 + \lambda h \sigma_0) \leq 0.$$

Since the left side is a convex function in λ, it is equivalent to check only for the extreme values

$$\begin{aligned} (\rho_1 + \mu h \sigma_1)^2 &\leq 4(\rho_0 + \mu h \sigma_0), \\ (\rho_1 + L h \sigma_1)^2 &\leq 4(\rho_0 + L h \sigma_0). \end{aligned}$$

As roots are complex conjugates,

$$|r_\pm(\lambda)|^2 = |c| = |\rho_0 + \lambda h \sigma_0|.$$

Because this one-dimensional function is convex in λ, the maximum is attained for an extreme value of λ,

$$\max_{\lambda \in [\mu, L]} |r_\pm(\lambda)|^2 = \max \left\{ \rho_0 + \mu h \sigma_0,\ \rho_0 + L h \sigma_0 \right\},$$

which is the desired result. ■

The next step is to minimize the largest modulus defined in (2.13) in the coefficients ρ_i and σ_i to get the best rate of convergence, assuming the roots are complex. We will not develop the case where the roots are real because this leads to weaker results.

2.4.3 Design of optimal two-step method for quadratics

We now have all ingredients to build a two-step method for which the sequence x_k converges quickly to x^* for quadratic functions. We need to solve the following problem,

$$\begin{array}{ll} \text{minimize} & \max \left\{ \rho_0 + \mu h \sigma_0,\ \rho_0 + L h \sigma_0 \right\} \\ \text{s.t.} & (\rho_0, \rho_1, \sigma_1) \in \mathcal{L} \\ & (\rho_1 + \mu h \sigma_1)^2 \leq 4(\rho_0 + \mu h \sigma_0) \\ & (\rho_1 + L h \sigma_1)^2 \leq 4(\rho_0 + L h \sigma_0), \end{array}$$

in the variables $\rho_0, \rho_1, \sigma_0, \sigma_1, h > 0$, where $\mathcal{L}$ is defined in (2.12). If we use the equality constraints in (2.12) and make the following change of variables,

$$\begin{cases} \hat{h} &= h(1 - \rho_0), \\ c_\mu &= \rho_0 + \mu h \sigma_0, \\ c_L &= \rho_0 + L h \sigma_0, \end{cases} \tag{2.14}$$

the problem becomes, for fixed $\hat{h}$,

$$\begin{array}{ll} \text{minimize} & \max\left\{c_\mu,\ c_L\right\} \\ \text{s.t.} & (-1 - c_\mu + \mu\hat{h})^2 \leq 4c_\mu \\ & (-1 - c_L + L\hat{h})^2 \leq 4c_L \\ & |Lc_\mu - \mu c_L| < |L - \mu|, \end{array}$$

in the variables c_μ, c_L. In that case, the optimal solution is given by

$$c_\mu^* = \left(1 - \sqrt{\mu\hat{h}}\right)^2, \quad c_L^* = \left(1 - \sqrt{L\hat{h}}\right)^2, \tag{2.15}$$

obtained by tightening the two first inequalities, for $\hat{h} \in]0, \frac{(1+\mu/L)^2}{L}[$ such that the last inequality is satisfied. Now if we fix $\hat{h}$ we can recover an optimal two step linear method defined by (ρ, σ) and an optimal step size h by using the equations in (2.14). We will use the following quantity

$$\beta \triangleq \frac{1 - \sqrt{\mu/L}}{1 + \sqrt{\mu/L}}. \tag{2.16}$$

A suboptimal two-step method. We can fix $\hat{h} = 1/L$ for example. All computations done, the parameters of this two-step method, called method $\mathcal{M}_1$, are

$$\mathcal{M}_1 = \begin{cases} \rho(z) & = \beta - (1+\beta)z + z^2, \\ \sigma(z) & = -\beta(1-\beta) + (1-\beta^2)z, \\ h & = \frac{1}{L(1-\beta)}, \end{cases} \tag{2.17}$$

and its largest modulus root (2.13) is given by

$$\text{rate}(\mathcal{M}_1) = \sqrt{\max\{c_\mu,\ c_L\}} = \sqrt{c_\mu} = 1 - \sqrt{\mu/L}.$$

Optimal two-step method for quadratics. We can compute the optimal $\hat{h}$ which minimizes the maximum of the two roots c_μ^* and c_L^* defined in (2.15). The solution is simply the one which balances the two terms in the maximum:

$$\hat{h}^* = \frac{(1+\beta)^2}{L} \quad \Rightarrow \quad c_\mu^* = c_L^*.$$

This choice of $\hat{h}$ leads to the method $\mathcal{M}_2$, described by

$$\mathcal{M}_2 = \begin{cases} \rho(z) & = \beta^2 - (1+\beta^2)z + z^2, \\ \sigma(z) & = (1-\beta^2)z, \\ h & = \frac{1}{\sqrt{\mu L}}, \end{cases} \tag{2.18}$$

with the rate of convergence

$$\text{rate}(\mathcal{M}_2) = \sqrt{c_\mu} = \sqrt{c_L} = \beta < \text{rate}(\mathcal{M}_1).$$

We will now see that methods $\mathcal{M}_1$ and $\mathcal{M}_2$ are actually related to Nesterov's method and Polyak's heavy ball algorithms.

2.5 On the link between integration and optimization

In the previous section, we derived a family of linear multi-step methods, parametrized by $\hat{h}$. We will now compare these methods to common optimization algorithms used to minimize L-smooth, μ-strongly convex functions.

2.5.1 Polyak's heavy ball method

The heavy ball method was proposed by Polyak [59]. It adds a momentum term to the gradient step

$$x_{k+2} = x_{k+1} - c_1 \nabla f(x_{k+1}) + c_2(x_{k+1} - x_k),$$

where $c_1 = (1-\beta^2)/\sqrt{\mu L}$ and $c_2 = \beta^2$ where β is defined in (2.16). We can organize the terms in the sequence to match the general structure of linear multi-step methods, to get

$$\beta^2 x_k - (1+\beta^2)x_{k+1} + x_{k+2} = (1-\beta^2)/\sqrt{\mu L}\left(-\nabla f(x_{k+1})\right).$$

We easily identify $\rho(z) = \beta^2 - (1+\beta^2)z + z^2$ and $h\sigma(z) = (1-\beta^2)/\sqrt{\mu L}z$. To extract h, we will assume that the method is consistent (see conditions (2.8)), which means

$$\begin{aligned} \rho(1) &= 0 && \text{Always satisfied} \\ h\rho'(1) &= h\sigma(1) && \Rightarrow\ h = \frac{1}{\sqrt{\mu L}}. \end{aligned}$$

All computations done, we can identify the "hidden" linear multi-step method as

$$\mathcal{M}_{\text{Polyak}} = \begin{cases} \rho(z) &= \beta^2 - (1+\beta^2)z + 1 \\ \sigma(z) &= (1-\beta^2)z \\ h &= \frac{1}{\sqrt{\mu L}}. \end{cases} \tag{2.19}$$

This shows that $\mathcal{M}_{\text{Polyak}} = \mathcal{M}_2$. In fact, this result was expected since Polyak's method is known to be optimal for quadratic functions. However, it is also known that Polyak's algorithm does not converge for a general smooth and strongly convex function [43].

2.5.2 Nesterov's accelerated gradient

Nesterov's accelerated method in its simplest form is described by two sequences x_k and y_k, with

$$\begin{aligned} y_{k+1} &= x_k - \frac{1}{L}\nabla f(x_k), \\ x_{k+1} &= y_{k+1} + \beta(y_{k+1} - y_k). \end{aligned}$$

As above, we will write Nesterov's accelerated gradient as a linear multi-step method by expanding y_k in the definition of x_k, to get

$$\beta x_k - (1+\beta)x_{k+1} + x_{k+2} = \frac{1}{L}\left(-\beta(-\nabla f(x_k)) + (1+\beta)(-\nabla f(x_{k+1}))\right).$$

Consistency of the method is then ensured by

$$\begin{aligned} \rho(1) &= 0 && \text{Always satisfied} \\ h\rho'(1) &= h\sigma(1) && \Rightarrow\ h = \frac{1}{L(1-\beta)}. \end{aligned}$$

After identification,

$$\mathcal{M}_{\text{Nest}} = \begin{cases} \rho(z) &= \beta - (1+\beta)z + z^2, \\ \sigma(z) &= -\beta(1-\beta) + (1-\beta^2)z, \\ h &= \frac{1}{L(1-\beta)}, \end{cases}$$

which means that $\mathcal{M}_1 = \mathcal{M}_{\text{Nest}}$.

2.5.3 Nesterov's method interpretation as a faster stable integration method

Pushing the analysis a little bit further, we can show *why* Nesterov's algorithm is faster than gradient method. There is of course a complete proof of its rate of convergence [55], even using the argument of differential equations [78, 79], but we take a more intuitive approach here. The key parameter is the step size h. If we compare it with the one in classical gradient method, Nesterov's method uses a step size which is $(1-\beta)^{-1} \approx \sqrt{L/\mu}$ larger.

Recall that, in continuous time, we have seen the rate of convergence of $x(t)$ to x^* given by

$$f(x(t)) - f(x^*) \leq e^{-2\mu t}(f(x_0) - f(x^*)).$$

The gradient method tries to approximate $x(t)$ using Euler approximation with step size $h = 1/L$, which means $x_k^{(\text{grad})} \approx x(k/L)$, so

$$f(x_k^{(\text{grad})}) - f(x^*) \approx f(x(k/L)) - f(x^*) \leq (f(x_0) - f(x^*))e^{-2k\frac{\mu}{L}}.$$

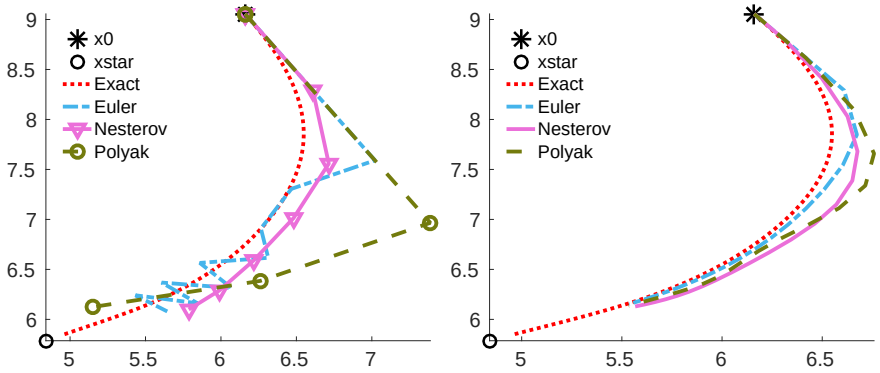


Figure 2.1: Integration of a (Linear ODE) (which corresponds to the minimization of a quadratic function) using Euler's, Nesterov's and Polyak's methods between $[0, t_{\max}]$. On the left, the optimal step size is used. Because Polyak's algorithm is the one with the biggest step size, it needs less iterations than Nesterov or Euler to approximate $x(t_{\max})$. On the right side, we reduced the step size to $1/L$ for all methods. We clearly observe that they all track the same ODE: the gradient flow.

However, Nesterov's method has the step size

$$h_{\text{Nest}} = \frac{1}{L(1-\beta)} = \frac{1+\sqrt{\mu/L}}{2\sqrt{\mu L}} \approx \frac{1}{\sqrt{4\mu L}} \qquad \text{which means} \quad x_k^{\text{nest}} \approx x\left(k/\sqrt{4\mu L}\right).$$

In that case, the estimated rate of convergence becomes

$$f(x_k^{\text{nest}}) - f(x^*) \approx f(x(k/\sqrt{4\mu L})) - f(x^*) \leq (f(x_0) - f(x^*))e^{-k\sqrt{\mu/L}},$$

which is approximatively the rate of convergence of Nesterov's algorithm in discrete time and we recover the accelerated rate in $\sqrt{\mu/L}$ versus μ/L for gradient descent. The accelerated method is more efficient because it integrates the gradient flow *faster* than simple gradient descent, making longer steps. A numerical simulation in Figure 2.1 makes this argument more visual. This intuitive argument is still valid for the convex counterpart of Nesterov's accelerated gradient.

2.6 Acceleration for convex functions

By matching the coefficients of Nesterov's method, we deduced the value of the step-size used for the integration of (Gradient Flow). Then, using the rate of convergence of $x(t)$ to x^*, we estimated the rate of convergence of Nesterov's method assuming $x_k \approx x(t_k)$. Here, we will do the same but without assuming strong convexity. However, the estimation of the rate of convergence in discrete time needs the one in continuous time, described by the following proposition.

Proposition 2.6.1. *Let f be L-smooth convex function, x^* one of its minimizers and $x(t)$ be*

the solution of (Gradient Flow). *Then*

$$f(x(t)) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{t + (2/L)}. \tag{2.20}$$

Proof. Let $\mathcal{L}(x(t)) = f(x(t)) - f(x^*)$. We notice that $\nabla \mathcal{L}(x(t)) = \nabla f(x(t))$ and $\mathrm{d}\mathcal{L}(x(t))/\,\mathrm{d}t = -\|\nabla f(x(t))\|^2$. Since f is convex,

$$\begin{aligned} \mathcal{L}(x(t)) &\leq \langle \nabla f(x(t)), x(t) - x^* \rangle \\ &\leq \|\nabla f(x(t))\| \|x(t) - x^*\|. \end{aligned}$$

By consequence,

$$-\|\nabla f(x(t))\|^2 \leq -\frac{\mathcal{L}(x(t))^2}{\|x(t) - x^*\|^2} \leq -\frac{\mathcal{L}(x(t))^2}{\|x_0 - x^*\|^2}. \tag{2.21}$$

The last inequality comes from the fact that $\|x(t) - x^*\|$ decreases over time,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\|x(t) - x^*\|^2 &= 2\langle \dot{x}(t), x(t) - x^* \rangle, \\ &= -2\langle \nabla f(x(t)), x(t) - x^* \rangle, \\ &\leq 0 \quad \text{since } f \text{ is convex.} \end{aligned}$$

From (2.21), we deduce the differential inequality

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathcal{L}(x(t)) \leq -\frac{\mathcal{L}(x(t))^2}{\|x_0 - x^*\|^2}.$$

The solution is obtained by integration,

$$\int_0^t \frac{\mathrm{d}\mathcal{L}(x(\tau))/\,\mathrm{d}\tau}{\mathcal{L}(x(\tau))^2}\,\mathrm{d}\tau \leq \int_0^t \frac{-1}{\|x_0 - x^*\|^2}.$$

The general solution is thus

$$\mathcal{L}(x(t)) \leq \frac{1}{\frac{t}{\|x_0 - x^*\|^2} + C},$$

for some constant $C > 0$. Since the inequality is valid for all time $t \geq 0$, the following condition on C,

$$\mathcal{L}(x(t)) \leq \frac{1}{\frac{t}{\|x_0 - x^*\|^2} + C} \leq \frac{1}{C} \quad \text{for} \quad t \geq 0,$$

is sufficient. Since $\mathcal{L}(x(t)) = f(x(t)) - f(x^*)$, setting $C = \frac{1}{f(x_0) - f(x^*)}$ satisfies the above inequality. Using smoothness of f,

$$f(x_0) - f(x^*) \leq \frac{L}{2}\|x_0 - x^*\|^2,$$

we get the desired result. ■

Assume we use Euler's method with step size $h = \frac{1}{L}$, the estimated rate of convergence will be

$$f(x_k) - f(x^*) \approx f(x(kh)) - f(x^*) \leq \frac{L\|x_0 - x^*\|^2}{k+2},$$

which is close to the rate of convergence of the classical gradient method for convex function. Now, consider Nesterov's method for minimizing a smooth and convex function f:

$$\begin{aligned} y_{k+1} &= x_k - \frac{1}{L}\nabla f(x_k) \\ x_{k+1} &= -\beta_k x_k + (1+\beta_k)x_{k+1}, \end{aligned}$$

where $\beta_k \approx \frac{k-2}{k+1}$. If we expand everything, we get after rearrangement,

$$\beta_k x_{k-1} - (1+\beta_k)x_k + x_{k+1} = \frac{1}{L}\left(\beta_k(-\nabla f(x_{k-1})) - (1+\beta_k)(-\nabla f(x_k))\right).$$

In other terms, we have an expression of the form $\rho_k(E)x_k = h_k\sigma_k(E)(-\nabla f(x_k))$. We can identify h if we assume the method consistent, which means

$$\begin{aligned} \rho(1) &= 0 \qquad\qquad \text{Always satisfied} \\ h_k\rho_k'(1) &= h_k\sigma_k(1) \qquad \Rightarrow\ h_k = \frac{1}{L(1-\beta_{k+1})} = \frac{(k+2)}{3L}. \end{aligned}$$

We can estimate, the rate of convergence of Nesterov's method using (2.20). Since $x_k \approx x(t_k)$,

$$x_k \approx x\big(\textstyle\sum_{i=0}^{k} h_i\big) \approx x\big(\frac{k^2}{6L}\big).$$

In terms of convergence to the optimal value,

$$f(x_k) - f(x^*) \approx f(x(t_k) - f(x^*) \leq \frac{6L\|x_0 - x^*\|^2}{k^2+12},$$

which is close to the bound from Nesterov [55]. Again, because the step-size of Nesterov's algorithm is larger (while keeping a stable sequence), we converge faster than the Euler's method.

2.7 Proximal algorithms and implicit integration methods

We present here links between proximal algorithms and implicit numerical methods that integrate the gradient flow equation. We begin with Euler's implicit method that corresponds to the proximal point algorithm.

2.7.1 Euler's implicit method and proximal point algorithm

We saw in Section 2.3.1 that Euler's explicit method used the Taylor expansion of the solution $x(t)$ of the (ODE) at the current point. The implicit version uses the Taylor expansion at the

next point which reads

$$x(t) = x(t+h) - h\dot{x}(t+h) + O(h^2).$$

If $t = kh$, by neglecting the second order term we get implicit Euler's method,

$$x_{k+1} = x_k + hg(x_{k+1}). \tag{2.22}$$

This recurrent equation requires to solve an implicit equation at each step that may be costly. However it provides in some cases better stability than the explicit version, especially when the differential. This is generally the case for implicit methods (see Süli and Mayers [75] for further details on implicit methods).

Now assume that g comes from a potential $-f$ such that we are integrating (Gradient Flow). Solving the implicit equation (2.22) is equivalent to compute the proximal operator of f defined as

$$\mathbf{prox}_{f,h}(x) = \underset{z}{\operatorname{argmin}} \frac{1}{2}\|z - x\|_2^2 + hf(z). \tag{2.23}$$

This can be easily verified by checking the first-order optimality conditions of the minimization problem. Euler's implicit method applied to (Gradient Flow) reads then

$$x_{k+1} = \mathbf{prox}_{f,h}(x_k),$$

where we recognize the proximal point algorithm [62].

We present now Mixed ODE that corresponds to composite optimization problems.

2.7.2 Implicit Explicit methods and proximal gradient descent

In numerical analysis, it is common to consider the differential equation

$$\dot{x} = g(x) + \omega(x), \tag{Mixed ODE}$$

where $g(x)$ is considered as the "non-stiff" part of the problem and ω the stiff one, where stiffness may be assimilated to bad conditioning [5, 27]. Usually, we assume ω integrable using an implicit method. If ω derives from a potential $-\Omega$ (meaning $\omega = -\nabla\Omega$), this is equivalent to assume that the proximal operator of Ω defined in (2.23) can be computed exactly.

We approximate the solution of (Mixed ODE) using IMplicit-EXplicit schemes (IMEX). In our case, we will focus on the following multi-step based IMEX scheme,

$$\rho(E)x_k = h\big(\sigma(E)g(x_k) + \gamma(E)\omega(x_k)\big),$$

where ρ, σ and γ are polynomials of degrees s, $s-1$ (the explicit part) and s respectively and ρ

is monic. It means that, at each iteration, we need to solve, in x_{k+s},

$$x_{k+s} = \sum_{i=0}^{s-1} \underbrace{\left(-\rho_i x_{k+i} + \sigma_i h g(x_{k+i}) + \gamma_i h \omega(x_{k+i})\right)}_{\text{known}} + \gamma_s \omega(x_{k+s}).$$

In terms of optimization the mixed ODE corresponds to composite minimization problems of the form

$$\text{minimize} \quad f(x) + \Omega(x), \tag{2.24}$$

where f, Ω are convex and Ω has a computable proximal operator. We can link IMEX schemes with many optimization algorithms which use the proximal operator, such as proximal gradient method, FISTA or Nesterov's method. For example, proximal gradient is written

$$\begin{aligned} y_{k+1} &= x_k - h\nabla f(x_k) \\ x_{k+1} &= \mathbf{prox}_{h\Omega}(y_{k+1}). \end{aligned}$$

After expansion, we get

$$x_{k+1} = y_{k+1} - h\nabla\Omega(x_{k+1}) = x_k + hg(x_k) + h\omega(x_{k+1}),$$

which corresponds to the IMEX method with polynomials

$$\rho(z) = -1 + z, \quad \sigma(z) = 1, \quad \gamma(z) = z.$$

However, for Fista and Nesterov's method, we need to use a variant of linear multi-step algorithms, called *one leg* methods [17, 81]. Instead of combining the gradients, the idea is to compute g at a linear combination of the previous points, i.e.

$$\rho(E)x_k = h\left(g(\sigma(E)x_k) + \omega(\gamma(E)x_k)\right).$$

Their analysis (convergence, consistency, interpretation of h, etc...) is slightly different from linear multi-step method, so we will not go into details in this chapter, but the correspondence still holds.

2.7.3 Non-smooth gradient flow

In the last subsection we assumed that ω comes from a potential. However in the optimization literature, composite problems have a smooth convex part and a non-smooth sub-differentiable convex part which prevents us from interpreting the problem with the gradient flow ODE. Non-smooth convex optimization problems can be treated with differential inclusions (see [10] for recent results on it)

$$\dot{x}(t) + \partial f(x(t)) \ni 0,$$

where f is a sub-differentiable function whose sub-differential at x is written $\partial f(x)$. Composite problems (2.24) can then be seen as the discretization of the differential inclusion

$$\dot{x}(t) + \nabla f(x(t)) + \partial\Omega(x(t)) \ni 0.$$

2.8 Mirror gradient descent and non-Euclidean gradient flow

In many optimization problems, it is common to replace the Euclidean geometry with a distance-generating function called $d(x)$, with the associated Bregman divergence

$$\mathcal{B}_d(x, y) = d(x) - d(y) - \langle \nabla d(y), x - y \rangle,$$

with d strongly-convex and lower semi-continuous. To take into account this geometry we consider the Non-Euclidean Gradient Flow [41]

$$\begin{aligned} \dot{y}(t) &= -\nabla f\left(x(t)\right) \\ x(t) &= \nabla d^*(y(t)) \\ x(0) &= x_0,\ y(0) = \nabla d(x_0). \end{aligned} \tag{NEGF}$$

Here ∇d maps primal variables to dual ones and, as d is strongly convex, $(\nabla d)^{-1} = \nabla d^*$, where d^* is the Fenchel conjugate of d. In fact, we can write (NEGF) using only one variable y, but this formulation has the advantage to exhibit both primal and dual variables $x(t)$ and $y(t)$. Applying the forward Euler's explicit method we get the following recurrent equation

$$y_{k+1} - y_k = -h\nabla f(x_k), \quad x_{k+1} = \nabla d^* y_{k+1}.$$

Now consider the mirror gradient scheme :

$$x_{k+1} = \operatorname*{argmin}_x \ h\langle \nabla f(x_k), x\rangle + \mathcal{B}_d(x, x_k).$$

First optimality condition reads

$$\nabla_x \left(h\langle \nabla f(x_k), x\rangle + \mathcal{B}_d(x, x_k)\right)\big|_{x=x_{k+1}} = h\nabla f(x_k) + \nabla d(x_{k+1}) - \nabla d(x_k) = 0$$

Using that $(\nabla d)^{-1} = \nabla d^*$ we get

$$h\nabla f(x_k) + y_{k+1} - y_k = 0, \quad x_{k+1} = \nabla d^* y_{k+1},$$

which is exactly Euler's explicit method defined in (NEGF).

2.9 Universal gradient descent and generalized gradient flow

Consider the Generalized Gradient Flow, which combines the ideas of (Mixed ODE) and (NEGF),

$$\begin{aligned} \dot{y}(t) &= -\nabla f(x(t)) - \nabla \Omega(x(t)) \\ x(t) &= \nabla d^*(y(t)) \\ x(0) &= x_0,\ y(0) = \nabla d(x_0). \end{aligned} \tag{GGF}$$

We can write its ODE counterpart, called the "Generalized ODE",

$$\begin{aligned} \dot{y}(t) &= g(x(t)) + \omega(x(t)) \\ x(t) &= \nabla d^*(y(t)), \\ x(0) &= x_0,\ y(0) = \nabla d(x_0). \end{aligned} \tag{GODE}$$

where $g = -\nabla f$, with f a smooth convex function, d a strongly convex and semi-continuous distance generating function and $\omega = -\partial\Omega$, where Ω is a simple convex function. If Ω is not differentiable we can consider the corresponding differential inclusion as presented in Section 2.7.3. Here we focus on (GODE) and (GGF) to highlight the links with integration methods. The discretization of this ODE is able to generate many algorithms in many different settings. For example, consider the IMEX explicit-implicit Euler's method,

$$\frac{y_{k+1} - y_k}{h} = g(x_k) + \omega(\nabla d^*(y_{k+1})), \quad x_{k+1} = \nabla d^*(y_{k+1}),$$

which can be decomposed into three steps,

$$\begin{aligned} z_{k+1} &= y_k + hg\left(x_k\right) && \text{(Gradient step in dual space)}, \\ y_{k+1} &= \mathbf{prox}_{h(\Omega \circ \nabla d^*)}\left(z_{k+1}\right) && \text{(Projection step in dual space)}, \\ x_{k+1} &= \nabla d^*(y_{k+1}) && \text{(Mapping back in primal space)}. \end{aligned} \tag{2.25}$$

Now consider the universal gradient method scheme presented by Nesterov [56]:

$$x_{k+1} = \arg\min_x \langle \nabla f(x_k), x - x_k \rangle + \Omega(x) + \mathcal{B}_d(x, x_k).$$

Again we can show that both recursions are the same: if we write the first optimality condition,

$$\begin{aligned} 0 &= \nabla_x \left(h\langle \nabla f(x_k), x - x_k \rangle + h\Omega(x) + \mathcal{B}(x, x_k) \right) \Big|_{x = x_{k+1}} \\ &= hg(x_k) + h\partial\Omega(x_{k+1}) + \nabla d(x_{k+1}) - \nabla d(x_k) \\ &= \underbrace{hg(x_k) - y_k}_{= z_{k+1}} + h\partial\Omega(\nabla d^*(y_{k+1})) - y_{k+1}. \end{aligned}$$

We thus need to solve the non-linear system of equations

$$y_{k+1} = z_{k+1} + h\partial\Omega(\nabla d^*(y_{k+1})),$$

which is equivalent to the projection step (2.25). Then we simply recover x_{k+1} by applying ∇d^* on y_{k+1}.

2.10 Conclusion and future works

We connected several optimization algorithms to multi-step integration methods in numerical analysis. By using the theory of linear multi-step methods on the basic gradient flow equation, we recover Polyak's and Nesterov's method using some optimality arguments. This provides an intuitive interpretation for the design of these methods, with optimal step sizes giving a direct explanation of the acceleration phenomenon. Our approach generalizes to more structured problems by introducing the appropriate integration method and/or looking at a generalized gradient flow equation that takes into account the geometry of the problem.

We described a simple interpretation of the acceleration phenomenon, but our analysis is still restricted to quadratic problems. The study of G-stability [16, 14] may generalize our approach to smooth strongly convex functions. For the non-strongly convex case, the dependence in k of Nesterov's algorithm makes its links with integration methods less clear.

Finally the study of the links between optimization and numerical methods may provide new algorithms for both fields. Conversely, Nesterov's algorithm may lead to new integration methods to integrate the gradient flow equation of a convex function.

Chapter 3

Regularized Nonlinear Acceleration

Abstract

We describe a generic convergence acceleration technique for iterative algorithms. Our scheme computes estimates of fixed-points from a nonlinear average of the iterates produced by the iterative method. The weights in this average are computed via a simple linear system, whose solution can be updated online. This acceleration scheme runs in parallel to the base algorithm, providing improved estimates of the solution on the fly, while the original method is running. In this chapter, we study different settings such as linear algorithms with arbitrary, potentially random, perturbations. We also propose an acceleration method when another linear combination with arbitrary coefficients is maintained at each iteration, which usually corresponds to algorithms with momentum or averaging.

Main references: The results in this chapter have been partially published in [64], [65] (NIPS 2016 and 2017) and [68] (under review NIPS 2018). A revised version of [64] is under review for the journal Mathematical Programming. In this thesis, in comparison with the published papers, we improved the theoretical analysis of the proposed algorithm, especially the convergence bounds.

3.1 Introduction

Suppose we run an iterative algorithm of the form

$$x_{i+1} = g(x_i)$$

which converges to a solution x^*. For example, such an algorithm is used in optimization for solving the minimization problem

$$\min_{x \in \mathbb{R}^d} f(x). \tag{3.1}$$

Here, we will focus on improving our estimates of the solution to problem (3.1) by tracking only the iterate sequence $\{x_i\}$ produced by an optimization algorithm, without any further calls to oracles on $g(x)$.

We will study methods for accelerating convergence of iterative algorithms, originally studied in from numerical analysis. In particular we propose a regularized version of Anderson's extrapolation [4]. This method uses the iterates produced by any (converging) iterative algorithm, and estimates the solution directly from this sequence, up to some regularity conditions. Our scheme is based on the idea behind Aitken's Δ^2-algorithm [1], generalized as the Shanks transform [71], whose recursive formulation is known as the ε-algorithm [80] (see e.g. [12, 72] for a survey). In a nutshell, these methods fit geometrical models to linearly converging scalar sequences, then extrapolate their limit from the fitted model. Our method can be viewed as an extension to vector sequences.

We start from a formulation of these techniques known as anderson Acceleration [4], Mešina's algorithm [47] or minimal polynomial extrapolation (MPE) [72, 73]. They use the minimal polynomial of the linear operator driving iterations to estimate the optimum by a nonlinear average of the iterates (i.e. computing a weighted average using weights which are nonlinear functions of the iterates).

Our contribution here is to regularize this procedure and produce explicit bounds on the distance to optimality by controlling stability, thus explicitly quantifying acceleration. We show that these extrapolation algorithms reach optimal performance (asymptotically) and describe several numerical examples where these stabilized estimates often speed up convergence by an order of magnitude. So far, for all the techniques cited above, no proofs of convergence of the estimates were given when the estimation process became unstable (for example when extrapolating iterates produced by nonlinear functions). Furthermore, the acceleration scheme can be run in parallel with the original algorithm, providing improved estimates of the solution on the fly, while the original method is progressing, so its numerical complexity is marginal.

Notations For the sake of simplicity, throughout the thesis we use $\|\cdot\|$ for the ℓ_2 norm of a vector, and its corresponding induced matrix norm

$$\|A\| = \max_{v:\|v\|=1} \|Av\|.$$

In this chapter, we distinguish perturbed iteration or operators with their "noiseless" counterpart using the tilde $\tilde{}$ notation. In particular, we distinguish two sequences of points generated by

$$x_{i+1} = g(x_i), \qquad \tilde{x}_{i+1} = \tilde{g}(\tilde{x}_i).$$

where g and $\tilde{g}$ share the same fixed-point, x^*. On the left, the function g is written

$$g(x) = G(x - x^*) + x^*$$

where G is a matrix. Its perturbed version reads

$$\tilde{g}(x_i) = g(x_i) + e_i$$

for some perturbation e_i.

3.2 Convergence Acceleration for Linear Algorithms

We begin by recalling the core arguments behind convergence acceleration. These ideas have taken various forms over time, known for example as *Anderson acceleration* [4], the *Eddy-Mesina method* [47, 23] and *minimal polynomial extrapolation* [15, 73]. The core idea behind these methods is to use a Taylor expansion of

$$\tilde{x}_{i+1} = g(\tilde{x}_i) = G(x_i - x^*) + e_i \tag{3.2}$$

to approximate the fixed point iterations by a vector autoregressive model, then compute a weighted mean of the iterates x_i to produce a better estimate of the (unique) limit x^*. In this section, we will study the rate of convergence of these algorithms when g is linear, i.e., when x_{i+1} is generated with the *linear fixed point iteration*

$$x_{i+1} = g(x_i) = G(x_i - x^*) + x^*, \tag{LFPI}$$

where we assume G to be symmetric, positive semi-definite and $\|G\| < (1 - \kappa)$, where $\kappa < 1$. When optimizing a quadratic function using the gradient method, κ is often refereed to be the inverse of the condition number.

This model corresponds, for example, to a linearization of (3.2). Suppose $g(x)$ is differentiable and let G be the Jacobian of g evaluated at x^*. If we linearize (3.2) around x^* we obtain

$$x_{i+1} = g(x^*) + G(x_i - x^*) + O(\|x_i - x^*\|^2), \quad \text{for } i = 1, \dots, k.$$

By neglecting the second order term, and because $g(x^*) = x^*$, we obtain the *linear fixed-point iteration* (LFPI).

Because $\|G\|_2 \leq (1 - \kappa) < 1$, the iterates x_i converge to x^* at a linear rate, with

$$\|x_i - x^*\| \leq (1 - \kappa)\|x_{i-1} - x^*\| \leq (1 - \kappa)^i \|x_0 - x^*\|,$$

where $\|\cdot\|$ stands for the Euclidean norm. Similarly, for linear g, the residual

$$r_i \triangleq g(x_i) - x_i \quad (= x_{i+1} - x_i) \tag{3.3}$$

converges with the same rate since it also follow the autoregressive process

$$\begin{aligned} r_i &= (x_{i+1} - x^*) - (x_i - x^*), \\ &= G\left((x_i - x^*) - (x_{i-1} - x^*)\right), \\ &= Gr_{i-1}, \end{aligned} \tag{3.4}$$

so we easily deduce that

$$\|r_i\| \leq \|G\|\|r_{i-1}\| \leq (1-\kappa)^i \|r_0\|.$$

We will now see how to improve convergence rates using a linear combination of the previous iterates. Suppose we run N iterations of (LFPI). We will see that a good linear combination called *extrapolation*,

$$x_{\text{extr}} = \sum_{i=0}^{N} c_i x_i, \tag{3.5}$$

can minimize the residual (3.3). In other words, we have to find c^* which satisfies

$$c^* = \underset{c}{\operatorname{argmin}} \|g(x_{\text{extr}}) - x_{\text{extr}}\|. \tag{3.6}$$

The next proposition shows that minimizing the combination of residuals (3.3) leads to the same result, if the coefficients c sum to one.

Proposition 3.2.1. *Let c^* be the solution of* (3.6) *when we ensure $\mathbf{1}^T c = 1$. Then c^* can be also recovered by minimizing the combination of residuals* (3.3)*, i.e.,*

$$c^* = \underset{c:c^T 1=1}{\operatorname{argmin}} \left\| \sum_{i=0}^{N} c_i r_i \right\|. \tag{3.7}$$

In particular, this implies

$$g(x_{extr}) - x_{extr} = \sum_{i=0}^{N} c_i^* r_i. \tag{3.8}$$

Proof. Equation (3.6) can be expanded using (3.5) and (LFPI),

$$g(x_{\text{extr}}) - x_{\text{extr}} = G\left(\sum_{i=0}^{N} c_i x_i - x^*\right) + x^* - \sum_{i=0}^{N} c_i x_i.$$

Since $\sum_{i=0}^{N} c_i = 1$, we can factor the sum,

$$g\left(x_{\text{extr}}\right) - x_{\text{extr}} = \sum_{i=0}^{N} c_i \Big(\underbrace{G\left(x_i - x^*\right) + x^*}_{=g(x_i)} - x_i \Big).$$

By definition of the residual (3.3), we obtain

$$g\left(x_{\text{extr}}\right) - x_{\text{extr}} = \sum_{i=0}^{N} c_i r_i.$$

This conclude the proof, since the objective functions of (3.6) and (3.7) are equal. ■

We can now minimize the residual of the extrapolation x_{extr}. It suffices to minimize the combination of the residuals r_i with coefficients c_i, then use these coefficients to combine the points x_i of the sequence. The next theorem gives an upper bound on the residual norm of extrapolation given some assumptions on the spectrum of the matrix G in (LFPI).

Theorem 3.2.2. ***(Scieur, d'Aspremont and Bach [64]).*** *Suppose the iterates x_i for $i = 0, \dots, N$ are computed using* (LFPI)*, with G symmetric positive semi-definite, satisfying $\|G\| \leq (1 - \kappa)$ for $\kappa < 1$. Let x^* be the unique fixed point of g and m be the number of distinct eigenvalues of G. Then, the ℓ_2 norm of the residual of the extrapolation x_{extr} (3.5) is bounded by*

$$\left\|g(x_{extr}) - x_{extr}\right\|_2 \leq \begin{cases} \dfrac{2\beta^N}{1+\beta^{2N}} \left\|r_0\right\|_2 & \text{if } N < m \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

where

$$\beta = \frac{1-\sqrt{\kappa}}{1+\sqrt{\kappa}} < 1. \tag{3.10}$$

Proof. First, we rewrite the residual of x_{extr} using polynomials. In view of Proposition 3.2.1 and equation (3.4),

$$g(x_{\text{extr}}) - x_{\text{extr}} = \sum_{i=0}^{N} c_i r_i = \underbrace{\sum_{i=0}^{N} c_i G^i}_{=p(G)} r_0.$$

The term $p(G)$ is a polynomial in the matrix G, with coefficients c_i. In this proof, we bound the value

$$\min_{p \in \mathbb{R}_N[x]: p(1)=1} \|p(G) r_0\| = \min_{c: \mathbf{1}^T c = 1} \|g(x_{\text{extr}}) - x_{\text{extr}}\|,$$

where $\mathbb{R}_N[x]$ is the linear space of polynomials of degree at most N.

Because G is symmetric, it admits the eigenvalue decomposition

$$G = Q^* \Lambda Q$$

where Λ is the diagonal matrix of eigenvalues $\{\lambda_i,\ i = 1, \ldots, m\}$, and Q is an orthogonal matrix. For any polynomial $p \in \mathbb{R}_N[x]$ (the space of polynomials of degree at most N), we have

$$\begin{aligned} \|p(G)r_0\| &= \|Q^* p(\Lambda) Q r_0\| \\ &\leq \|p(\Lambda)\|\ \|r_0\| \\ &= \max_{i=1,\ldots,m} |p(\lambda_i)|\ \|r_0\|. \end{aligned}$$

There are two cases to analyze here, which are $N < m$ and $N \geq m$.

Case $N \geq m$. Using Cayley-Hamilton theorem, if $p(x)$ is the characteristic polynomial of G, then $p(G) = p(\Lambda) = 0$. By assumption none of the λ_i is equal to 1 (we assumed $\lambda_i \in [0, 1-\kappa]$ with $0 < \kappa < 1$) so we can normalize p so that $p(1) = 1$ and $p(\lambda_i) = 0$ for all $i = 1, \ldots, m$.

Case $N < m$. We have, for any polynomial p,

$$\max_{i=1,\ldots,m} |p(\lambda_i)| \leq \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |p(\lambda)|$$

Because $0 \preceq G \preceq 1 - \kappa$, we have $0 \leq \lambda_i \leq 1 - \kappa$, so the error bound becomes

$$\min_{\{p \in \mathbb{R}_N[x]:\, p(1)=1\}} \|p(G)(x_0 - x^*)\|_2 \leq \min_{\{p \in \mathbb{R}_N[x]:\, p(1)=1\}}\ \max_{\lambda \in [0, 1-\kappa]} |p(\lambda)|\ \|(x_0 - x^*)\|_2 \tag{3.11}$$

where the right hand side involves a minmax problem, explicitly solved using Chebyshev's polynomials. Let $C_N(x)$ be the Chebyshev polynomial of degree N. By definition, C_N is a monic polynomial (i.e. a polynomial whose leading coefficient is one) solving

$$C_N(x) \triangleq \operatorname*{argmin}_{\{p \in \mathbb{R}_N[x]:\, p=1\}}\ \max_{x \in [-1,1]} |p(x)|.$$

Golub and Varga [30] use a variant of $C_N(x)$ to solve the problem in (3.11), whose solution is a rescaled Chebyshev polynomial given by

$$T_N(x, \kappa) = \frac{C_N(t(x,\kappa))}{C_N(t(1,\kappa))}, \quad \text{where} \quad t(x,\kappa) = \frac{2x - (1-\kappa)}{1-\kappa}, \tag{3.12}$$

where $t(x, \kappa)$ is simply a linear mapping from interval $[0, 1-\kappa]$ to $[-1, 1]$. Moreover, they show that the maximal value is reached at the largest value of the interval,

$$\min_{\{p \in \mathbb{R}_N[x]:\, p(1)=1\}}\ \max_{\lambda \in [0,1-\kappa]} |p(\lambda)| = \max_{\lambda \in [0,1-\kappa]} |T_N(\lambda, \kappa)| = |T_N(1-\kappa, \kappa)| = \frac{2\beta^N}{1 + \beta^{2N}}, \tag{3.13}$$

where β is given by

$$\beta = \frac{1 - \sqrt{\kappa}}{1 + \sqrt{\kappa}} < 1 - \kappa < 1.$$

Injecting the result of (3.13) in (3.11) yields the desired result. ■

Since in (3.7) we involve r_N in the computation of the coefficients c^*, it implies we have computed x_{N+1} because $r_N = x_{i+1} - x_i$. However, x_{extr} does not contain the information of the last point of the sequence because $x_{\text{extr}} = \sum_{i=0}^{N} c_i^* x_i$. This can be considered as a waste of information, since x_{N+1} is more likely closer to the optimum x^*. The following corollary shows that using the same coefficients c_i we can incorporate x_{N+1} in the extrapolation process, and recover $g(x_{\text{extr}})$ instead of x_{extr}.

Corollary 3.2.3. *If g is a linear function, we have*

$$g(x_{extr}) = \sum_{i=1}^{N+1} c_i^* x_i.$$

If we use this point instead of x_{extr}, we have the following bound,

$$\|g\left(g(x_{extr})\right) - g(x_{extr})\| \leq (1-\kappa)\|g(x_{extr}) - x_{extr}\|.$$

Proof. By linearity of g, and because $\mathbf{1}^T c^* = 1$, we have

$$\sum_{i=1}^{N+1} c_i^* x_i = \sum_{i=0}^{N} c_i^* g(x_i) = g\left(\sum_{i=0}^{N} c_i^* x_i\right) = g(x_{\text{extr}}).$$

■

The proof of Theorem 3.2.2 suggests $T_N(x,\kappa)$ is a good universal solution for the convergence acceleration problem. For illustration, we plot both $T_3(x,\kappa)$ and $T_5(x,\kappa)$ for $x \in [0,1]$ and $\kappa = 0.15$ in Figure 3.1. This solution is called the *Chebyshev semi-iterative method* in [30] and was further studied by e.g. [51]. Combining the N iterates x_i using the coefficients c_i in $T_N(x,\kappa)$, we ensure

$$\|g(x_{\text{extr}}) - x_{\text{extr}}\| \leq (1-\sqrt{\kappa})^N r_0 \ll (1-\kappa)^N r_0,$$

which means convergence is indeed accelerated. However, this method has some key drawbacks. First, we need to know κ to form $T_N(x,\kappa)$, which is not always the case. For example, when g is in fact nonlinear, κ depends on the spectrum of the Jacobian at the optimum, which is clearly not observed. Second, the algorithm does not allow us to control the magnitude of the coefficients in the polynomial $T(x,\kappa)$, which has a strong impact on the stability of the algorithm in the presence of numerical errors, or when the iterates are generated by a non-linear function $\tilde{g}$.

Because of stability issues with Chebyshev acceleration, we focus instead on the method (3.7) which computes the c_i adaptively (which also has the same bound of convergence). We describe it formally in Algorithm 1. This acceleration algorithm is called nonlinear because the coefficients c_i vary with x_i. This method is also known as Anderson acceleration [4], the Eddy-Mesina algorithm [47, 23], minimal polynomial extrapolation [15], or reduced rank extrapolation

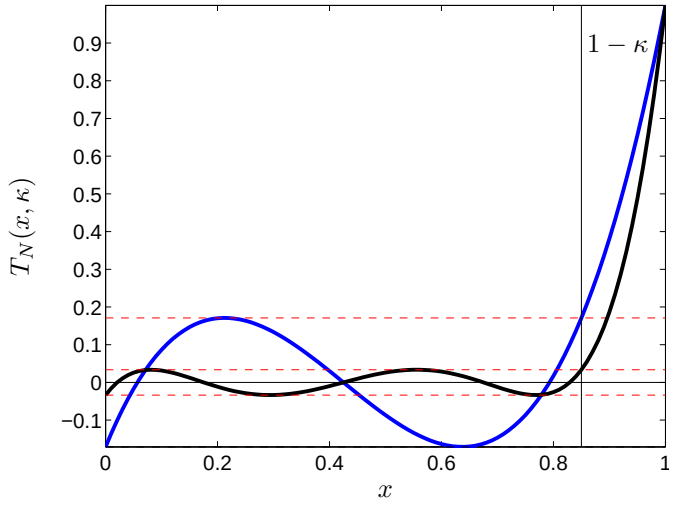


Figure 3.1: We plot both $T_3(x,\kappa)$ (blue) and $T_5(x,\kappa)$ (black) for $x \in [0,1]$ and $\sigma = 0.85$. The maximum value of the image of $[0, 1-\kappa]$ by T_N is clearly smaller when N grows, implying a better rate of convergence.

Algorithm 1 Nonlinear Acceleration of Convergence

Require: Iterates $x_0, x_1, \ldots, x_{N+1} \in \mathbb{R}^d$.

1: Form $R = [r_0, ..., r_N]$, where r_i is defined in (3.3).

2: Solve

$$c^* = \underset{\{c \in \mathbb{R}^{N+1}:\, c^T 1 = 1\}}{\operatorname{argmin}} \|Rc\|$$

3: Compute $x_{\text{extr}} = \sum_{i=0}^N c_i x_i$ and $g(x_{\text{extr}}) = \sum_{i=0}^N c_i x_{i+1}$.

Ensure: Residual $\|g(x_{\text{extr}}) - x_{\text{extr}}\|$ small.

[72, 73]. There are small variations between all these methods, which lie in the way they solve the minimization problem in (3.7).

In a sense, theses approaches are more statistical in nature. They assume an approximately linear model holds for iterations near the optimum, and estimate this model using the iterates. When working with scalar sequences, it has strong links with Wynn's algorithm [80], directly connected to the Levinson-Durbin algorithm [44, 22] used to solve Toeplitz systems recursively and fit autoregressive models (the Shanks transform solves Hankel rather than Toeplitz systems, but this is essentially the same problem [36]). The key difference in these extrapolation techniques is that estimating the autocovariance operator G is not required, as we only focus on the limit.

We see that Algorithm (1) needs to solve an optimization problem to find the c_i. Hopefully, there exists a simpler way to perform the computation. The next proposition gives us an explicit solution to step 2 in Algorithm 1, involving the inversion of a N-by-N matrix.

Proposition 3.2.4. *The explicit solution of the problem*

$$c^* = \underset{c^T 1=1}{\operatorname{argmin}} \|Rc\| \tag{3.14}$$

in the variable $c \in \mathbb{R}^N$, where R is a $d \times N$ matrix assumed to be of rank N, is given by

$$c^* = \frac{(R^T R)^{-1}\mathbf{1}}{\mathbf{1}^T (R^T R)^{-1}\mathbf{1}}. \tag{3.15}$$

Proof. Let μ be the dual variable of the equality constraint. Both c^* and μ^* should satisfy the KKT system

$$\begin{bmatrix} 2R^T R & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} c^* \\ \mu^* \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3.16}$$

This block matrix can be inverted explicitly, with

$$\begin{bmatrix} 2R^T R & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix}^{-1} = \frac{1}{\mathbf{1}^T (R^T R)^{-1}\mathbf{1}} \begin{bmatrix} \frac{1}{2}(R^T R)^{-1}\left(\mathbf{1}^T (R^T R)^{-1}\mathbf{1} I - \mathbf{1}\mathbf{1}^T (R^T R)^{-1}\right) & (R^T R)^{-1}\mathbf{1} \\ \mathbf{1}^T (R^T R)^{-1} & -2 \end{bmatrix}.$$

Using this inverse we easily solve the linear system, which gives the result in (3.15). ■

In practice, instead of computing the inverse of the matrix $R^T R$, we solve the linear system

$$R^T R z = \mathbf{1},$$

then set $c^* = z/(\mathbf{1}^T z)$. Of course, we need to assume the matrix invertible, and this likely holds until $N > m$ in Theorem 3.2.2. In the case where the matrix is singular, it is possible to show, using the pseudo-inverse of $R^T R$, that any solution of the system recovers exactly x^*.

The formula (3.14) is used in Anderson acceleration algorithm and in the Mesina method. Other algorithms usually force the last coefficient c_N to be equal to one, solve the remaining linear system, then normalize the vector. However, these alternative strategies are harder to analyze when the iterates are generated by a non-linear function g.

3.3 Regularized Nonlinear Acceleration of Convergence

So far, we have only considered linear functions g in (LFPI), without perturbations, when computing the iterates x_i. In general, the fixed-point iteration (3.2) is usually generated by a non-linear function g, thus inducing a second order error term compared to the dynamics in (LFPI). We can also assume each iteration is corrupted by a random noise. In this section, we will study the impact of perturbations on the acceleration algorithm and propose a new variant which is robust to noise.

3.3.1 Sensitivity Analysis

We now study the sensitivity of the acceleration algorithm to perturbations. Consider the following perturbed linear fixed point iteration

$$\tilde{x}_{i+1} - x^* \triangleq \tilde{g}(\tilde{x}_i) - x^* = G(\tilde{x}_i - x^*) + e_i \qquad \text{(Pert. LFPI)}$$

where e_i is the noise injected at iteration i. For now, we do not assume any structure on the noise, so e_i may come from nonlinearities of $\tilde{g}$, stochastic noise, roundoff error, *etc.* The iterates of this process will be compared to their noiseless counterpart,

$$x_{i+1} - x^* = g(x_i) - x^* = G(x_i - x^*),$$

with $x_0 = \tilde{x}_0$. We now apply our acceleration algorithm on the sequences x_i and $\tilde{x}_i$ and compare the results. We first form the residuals,

$$r_i = g(x_i) - x_i = x_{i+1} - x_i \qquad \text{and} \qquad \tilde{r}_i = \tilde{g}(\tilde{x}_i) - \tilde{x}_i = \tilde{x}_{i+1} - \tilde{x}_i.$$

Consider the matrices of residuals $R = [r_0, \ldots, r_N]$ and $\tilde{R} = [\tilde{r}_0, \ldots, \tilde{r}_N]$. We write P the *perturbation matrix* defined as

$$P \triangleq \tilde{R}^T\tilde{R} - R^TR. \tag{3.17}$$

The next proposition describes the sensitivity of Algorithm 1 using R and P.

Proposition 3.3.1. *Let the sequences x_i be generated by* (LFPI) *and $\tilde{x}_i$ by* (Pert. LFPI)*, with $x_0 = \tilde{x}_0$, with R and $\tilde{R}$ the residual matrices defined above with $\tilde{R}^T\tilde{R}$ invertible and P the perturbation matrix in* (3.17)*. Assume c^* and $\tilde{c}^*$ are computed using formula* (3.15) *with matrices R and $\tilde{R}$ respectively. Let*

$$\Delta c^* \triangleq \tilde{c}^* - c^*. \tag{3.18}$$

Then the norm of Δc^* is bounded by

$$\|\Delta c^*\| \leq \|P\|\|(\tilde{R}^T\tilde{R})^{-1}\|\|c^*\|. \tag{3.19}$$

Proof. We start with the sequence $\tilde{x}_i$. Let $\tilde{\mu}^*$ be the dual variable of the equality constraint of (3.14). Both $\tilde{c}^* = c^* + \Delta c^*$ and $\tilde{\mu}^* = \mu^* + \Delta\mu^*$ should satisfy the KKT system

$$\begin{bmatrix} 2\tilde{R}^T\tilde{R} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} \tilde{c}^* \\ \tilde{\mu}^* \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \Leftrightarrow \quad \begin{bmatrix} 2(R^TR + P) & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} c^* + \Delta c^* \\ \mu^* + \Delta\mu^* \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Indeed, using the definition of c^* and μ^* in (3.16),

$$\begin{aligned}
&\begin{bmatrix} 2(R^TR+P) & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} c^* + \Delta c^* \\ \mu^* + \Delta\mu^* \end{pmatrix} \\
= &\begin{bmatrix} 2R^TR & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} c^* \\ \mu^* \end{pmatrix} + \begin{bmatrix} 2R^TR & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} \Delta c^* \\ \Delta\mu^* \end{pmatrix} + \begin{bmatrix} 2P & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} c^* + \Delta c^* \\ \mu^* + \Delta\mu^* \end{pmatrix}, \\
= &\begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{bmatrix} 2R^TR & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} \Delta c^* \\ \Delta\mu^* \end{pmatrix} + \begin{bmatrix} 2P & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} c^* + \Delta c^* \\ \mu^* + \Delta\mu^* \end{pmatrix}.
\end{aligned}$$

With this simplification, the system becomes

$$\begin{bmatrix} 2R^TR & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} \Delta c^* \\ \Delta\mu^* \end{pmatrix} + \begin{bmatrix} 2P & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} c^* + \Delta c^* \\ \mu^* + \Delta\mu^* \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

It remains to isolate c^*,

$$\begin{bmatrix} 2(R^TR+P) & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{pmatrix} \Delta c^* \\ \Delta\mu^* \end{pmatrix} = \begin{pmatrix} 2Pc^* \\ 0 \end{pmatrix}.$$

The explicit solution in obtained by inverting the block matrix, and is written

$$\Delta c^* = \left(I - \frac{(R^TR+P)^{-1}\mathbf{1}\mathbf{1}^T}{\mathbf{1}^T(R^TR+P)^{-1}\mathbf{1}} \right) (R^TR+P)^{-1}Pc^*.$$

We can bound the norm of Δc^* by

$$\|\Delta c^*\| = \left\| I - \frac{(\tilde{R}^T\tilde{R})^{-1}\mathbf{1}\mathbf{1}^T}{\mathbf{1}^T(\tilde{R}^T\tilde{R})^{-1}\mathbf{1}} \right\| \|(\tilde{R}^T\tilde{R})^{-1}\|\|P\|\|c^*\|.$$

Because the first factor is the norm of a projector of rank N, its value is bounded by 1, so we get the desired result. ■

This proposition bounds the relative error on $\tilde{c}^*$ in comparison with c^*. We will see that the perturbation magnitude can be arbitrarily large, which is the key issue with the convergence results in [73, §7]. Even when $\|P\|$ is small, the term $\|(R^TR+P)^{-1}\|$ is problematic. Our problem here is the structure of the residuals matrix R,

$$R = [r_0, Gr_0, G^2r_0, ..., G^Nr_0],$$

which matches exactly the structure of *Krylov matrices*, i.e. square matrices K formed using a matrix M and a vector v, and computed as $K = [v, Mv, M^2v, ..., M^Nv]$. Tyrtyshnikov [77] showed that the condition number of Krylov matrices is lower bounded by a function which grows exponentially with N. Now, the error bound (3.19) contains the norm of the inverse of a

perturbed squared Krylov matrix, which makes the situation even worse. In other words, even if the perturbations are small, their impact on the solution can be arbitrarily large. Even in practical cases where N is small (for example, $N = 5$), $\tilde{R}^T\tilde{R}$ is usually a singular or nearly-singular matrix. This particular issue means the linear system $(R^TR)^{-1}\mathbf{1}$ in (3.15) needs to be regularized.

3.3.2 Regularized Nonlinear Acceleration

In this section, we will analyze the following acceleration algorithm, which uses Tikhonov regularization to solve the linear system in (3.15).

Algorithm 2 Regularized Nonlinear Acceleration (RNA)

Require: Iterates $\tilde{x}_0, \tilde{x}_1, ..., \tilde{x}_{N+1} \in \mathbb{R}^d$ produced by (Pert. LFPI), and a regularization parameter $\lambda > 0$.

1: Compute $\tilde{R} = [\tilde{r}_0, ..., \tilde{r}_N]$, where $\tilde{r}_i = \tilde{x}_{i+1} - \tilde{x}_i$

2: Solve

$$\tilde{c}^*_\lambda = \underset{c^T1=1}{\operatorname{argmin}} \|\tilde{R}c\|^2 + \lambda\|c\|^2,$$

or equivalently solve $(\tilde{R}^T\tilde{R} + \lambda I)z = \mathbf{1}$ then set $\tilde{c}^*_\lambda = z/\mathbf{1}^Tz$.

Ensure: Approximation of x^* computed as $\sum_{i=0}^{N}(\tilde{c}^*_\lambda)_i\tilde{x}_i$, or $\sum_{i=1}^{N+1}(\tilde{c}^*_\lambda)_i\tilde{x}_i$

Regularization controls the norm of the coefficients produced by the algorithm and reduces the impact of perturbations, as shown in the following proposition.

Proposition 3.3.2. *Consider the sequences x_i satisfying* (LFPI) *and $\tilde{x}_i$ satisfying* (Pert. LFPI) *with $x_0 = \tilde{x}_0$. Let c^*_λ and $\tilde{c}^*_\lambda$ the output of Algorithm 2 with parameter λ applied to x_i and $\tilde{x}_i$ respectively. Let R and $\tilde{R}$ the matrices of residuals and P be defined in* (3.17)*. Define $\Delta\tilde{c}^*_\lambda = \tilde{c}^*_\lambda - c^*_\lambda$. Then, we have the following bounds,*

$$\|\tilde{c}^*_\lambda\| \leq \sqrt{\frac{\lambda + \|\tilde{R}\|^2}{(N+1)\lambda}}, \tag{3.20}$$

$$\|\Delta\tilde{c}^*_\lambda\| \leq \frac{\|P\|}{\lambda}\|c^*_\lambda\|, \tag{3.21}$$

which control the stability of the solution $\tilde{c}^*_\lambda$.

Proof. Using the same proof technique of Propositions 3.2.4 and 3.3.1, we have

$$\tilde{c}^*_\lambda = \frac{(\tilde{R}^T\tilde{R} + \lambda I)^{-1}\mathbf{1}}{\mathbf{1}^T(\tilde{R}^T\tilde{R} + \lambda I)^{-1}\mathbf{1}}, \tag{3.22}$$

$$\Delta c^*_\lambda = \left(I - \frac{(\tilde{R}^T\tilde{R} + \lambda I)^{-1}\mathbf{1}\mathbf{1}^T}{\mathbf{1}^T(\tilde{R}^T\tilde{R} + \lambda I)^{-1}\mathbf{1}}\right)(\tilde{R}^T\tilde{R} + \lambda I)^{-1}Pc^*_\lambda. \tag{3.23}$$

We begin by the bound on $\tilde{c}^*_\lambda$. Indeed, with (3.22),

$$\begin{aligned}
\|\tilde{c}^*_\lambda\|^2 &= \frac{\mathbf{1}^T(\tilde{R}^T\tilde{R}+\lambda I)^{-2}\mathbf{1}}{(\mathbf{1}^T(\tilde{R}^T\tilde{R}+\lambda I)^{-1}\mathbf{1})^2}, \\
&\leq \frac{1}{N+1}\max_{\|v\|=1}\frac{v^T(\tilde{R}^T\tilde{R}+\lambda I)^{-2}v}{(v^T(\tilde{R}^T\tilde{R}+\lambda I)^{-1}v)^2}, \\
&= \frac{1}{N+1}\max_{\|v\|=1}\frac{\|(\tilde{R}^T\tilde{R}+\lambda I)^{-\frac{1}{2}}(\tilde{R}^T\tilde{R}+\lambda I)^{-\frac{1}{2}}v\|^2}{\|(\tilde{R}^T\tilde{R}+\lambda I)^{-\frac{1}{2}}v\|^4}, \\
&\leq \frac{1}{N+1}\|(\tilde{R}^T\tilde{R}+\lambda I)^{-\frac{1}{2}}\|^2\max_{\|v\|=1}\frac{1}{\|(\tilde{R}^T\tilde{R}+\lambda I)^{-\frac{1}{2}}v\|^2}, \\
&= \frac{1}{N+1}\|(\tilde{R}^T\tilde{R}+\lambda I)^{-\frac{1}{2}}\|^2\|(\tilde{R}^T\tilde{R}+\lambda I)^{\frac{1}{2}}\|^2.
\end{aligned}$$

The norm of the coefficients $\tilde{c}^*_\lambda$ are thus bounded by

$$\|\tilde{c}^*_\lambda\| \leq \sqrt{\frac{1}{N+1}\frac{\|\tilde{R}^T\tilde{R}\|+\lambda}{\lambda}} = \sqrt{\frac{\|\tilde{R}\|^2+\lambda}{(N+1)\lambda}}.$$

We will now bound $\|\Delta\tilde{c}_\lambda\|$. With equation (3.23),

$$\begin{aligned}
\|\Delta\tilde{c}_\lambda\| &= \left\|\left(I-\frac{(\tilde{R}^T\tilde{R}+\lambda I)^{-1}\mathbf{1}\mathbf{1}^T}{\mathbf{1}^T(\tilde{R}^T\tilde{R}+\lambda I)^{-1}\mathbf{1}}\right)(\tilde{R}^T\tilde{R}+\lambda I)^{-1}Pc^*_\lambda\right\|, \\
&\leq \left\|I-\frac{(\tilde{R}^T\tilde{R}+\lambda I)^{-1}\mathbf{1}\mathbf{1}^T}{\mathbf{1}^T(\tilde{R}^T\tilde{R}+\lambda I)^{-1}\mathbf{1}}\right\|\left\|(\tilde{R}^T\tilde{R}+\lambda I)^{-1}\right\|\|P\|\,\|c^*_\lambda\|, \\
&\leq \left\|(\tilde{R}^T\tilde{R}+\lambda I)^{-1}\right\|\|P\|\,\|c^*_\lambda\|,
\end{aligned}$$

where the last inequality is obtained by bounding the norm of a projector. Since $\tilde{R}^T\tilde{R} \succeq 0$, we have $(\tilde{R}^T\tilde{R}+\lambda I) \succeq \lambda I$, we get

$$\|\Delta\tilde{c}_\lambda\| \leq \frac{\|P\|}{\lambda}\|c^*_\lambda\|,$$

which is the desired result. ■

Regularization allows a better control of the impact of perturbations, but also changes the solution c^* into c^*_λ. The next section analyses the impact of regularization on the extrapolated solution when there are no perturbations.

3.3.3 Constrained Chebyshev Polynomial

The previous section shows that regularization is important to control the perturbations present in (Pert. LFPI). However, convergence analysis becomes more complicated in the perturbation-free case, and we introduce *constrained Chebyshev polynomials.*

Definition 3.3.3. *The constrained Chebyshev polynomial $C^*_\kappa(x, N, \gamma)$ of degree N, range $(1-\kappa)$ and norm constraint $\gamma > 1$ is defined as the solution of*

$$C^*_\kappa(x, N, \lambda) = \underset{C \in \mathbb{R}_N[x] \,:\, C(1)=1}{\operatorname{argmin}} \; \max_{x \in [0, 1-\kappa]} C(x), \qquad s.t. \;\; \|C\| \leq \frac{\gamma}{\sqrt{1+N}}$$

where $\|C\|$ corresponds to the ℓ_2 norm of the coefficients of polynomial C. We write the maximum value as

$$S_\kappa(N, \gamma) \triangleq \max_{x \in [0, 1-\kappa]} C^*_\kappa(x, N, \kappa). \tag{3.24}$$

Unfortunately, as far as we know, there exists no explicit bound for $S_\kappa(N, \gamma)$. However, it is still possible to compute this value using sum-of-squares.

Constrained Chebyshev Polynomials and SOS

We briefly recall basic results on Sum of Squares (SOS) polynomials and moment problems [53, 42, 58], which will allow us to formulate problem (3.24) as a (tractable) semidefinite program. A univariate polynomial is positive if and only if it is a sum of squares. Furthermore, if we let $m(x) = (1, x, \ldots, x^N)^T$ we have, for any $q(x) \in \mathbb{R}_{[2N]}$, the space of real polynomials of degree at most $2N$,

$$\begin{gathered} q(x) \geq 0, \text{ for all } x \in \mathbb{R} \\ \Updownarrow \\ q(x) = m(x)^T C m(x), \text{ for some } C \succeq 0, \end{gathered}$$

which means that checking if a polynomial is non-negative on the real line is equivalent to solving a linear matrix inequality (see e.g. [8, §4.2] for details). We can thus write the problem of computing the maximum of a polynomial over the real line as

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & t - p(x) = m(x)^T C m(x), \quad \text{for all } x \in \mathbb{R} \\ & C \succeq 0, \end{array} \tag{3.25}$$

which is a semidefinite program in the variables $p \in \mathbb{R}^{N+1}$, $C \in \mathbf{S}_{N+1}$ (the space of real positive-semidefinite matrices) and $t \in \mathbb{R}$, because the first contraint is equivalent to a set of linear equality constraints. Then, showing that $p(x) \geq 0$ on the segment $[0, 1-\kappa]$ is equivalent to showing that the rational fraction

$$p\left(\frac{(1-\kappa)x^2}{1+x^2}\right)$$

is non-negative on the real line, or equivalently, that the polynomial

$$(1+x^2)^N \; p\left(\frac{(1-\kappa)x^2}{1+x^2}\right)$$

is non-negative on the real line. Overall, this implies that problem (3.24) can be written

$$\begin{array}{lll} S_\kappa\left(N,\gamma\right) = & \min & t \\ & \text{s.t.} & t-(1+x^2)^{N+1}\left(\left(1-\frac{(1-\kappa)x^2}{1+x^2}\right) q\left(\frac{(1-\kappa)x^2}{1+x^2}\right)\right) = m(x)^T C m(x), \quad \text{for all } x\in\mathbb{R} \\ & & \mathbf{1}^T q = 1,\ \|q\|_2 \le \gamma,\ C \succeq 0, \end{array} \tag{3.26}$$

which is a semidefinite program in the variables $q \in \mathbb{R}^{N+1}$, $C \in \mathbf{S}_{N+2}$ and $t \in \mathbb{R}$.

Constrained Chebyshev Polynomials and Acceleration

Using $S_\kappa\left(N,\gamma\right)$ we can now bound the accuracy of the extrapolated point using the regularized algorithm. For simplicity, we consider here the *constrained* version of Algorithm 2, where the coefficients are computed using the norm constraint

$$c_\gamma^* = \underset{c:\mathbf{1}^T c=1}{\operatorname{argmin}} \|Rc\| \qquad \text{s.t. } \|c\| \le \frac{\gamma}{\sqrt{1+N}}.$$

Equivalently, $\tilde{c}_\gamma^*$ is computed using $\tilde{R}$ instead of R. This assumption is not restrictive, because we control the norm of the coefficients $\|c_\lambda^*\|$ (Proposition 3.3.2) thanks to the parameter λ. The next proposition shows the relation between γ and λ when we use Algorithm 2 on (Pert. LFPI).

Proposition 3.3.4. *Let the sequence $\tilde{x}_i$ generated from* (Pert. LFPI)*. Assume we want to use Algorithm 2 on this sequence, with $\|\tilde{c}_\lambda^*\| \le \frac{\gamma}{\sqrt{1+N}}$. In this case, λ should satisfies*

$$\lambda \ge \frac{\|\tilde{R}\|^2}{(\gamma^2-1)}. \tag{3.27}$$

Proof. Using the result from Proposition 3.3.2,

$$\|\tilde{c}_\lambda^*\| \le \frac{1}{\sqrt{N+1}}\sqrt{1+\frac{\|\tilde{R}\|^2}{\lambda}}$$

Since we want $\|\tilde{c}_\lambda^*\| \le \frac{\gamma}{\sqrt{1+N}}$, in the worst case,

$$\frac{1}{\sqrt{N+1}}\sqrt{1+\frac{\|\tilde{R}\|^2}{\lambda}} \le \frac{\gamma}{\sqrt{N+1}} \qquad \Leftrightarrow \qquad \lambda \ge \frac{\|\tilde{R}\|^2}{(\gamma^2-1)}.$$

■

The previous proposition also works when we apply the acceleration algorithm on (LFPI). We can now show the link between constrained Chebyshev polynomials and the accuracy of the extrapolation in the perturbation-free case.

Proposition 3.3.5. *Assume we have a sequence of points x_i generated by* (LFPI)*. Assume we use Algorithm 2, where the norm of the coefficients is bounded by $\frac{\gamma}{\sqrt{1+N}}$. Then, the residual norm of the extrapolation in the perturbation-free case is bounded by*

$$\left\| g\left(\sum_{i=0}^{N}(c_\gamma^*)_i x_i\right) - \sum_{i=0}^{N}(c_\gamma^*)_i x_i \right\| \leq \|r_0\| S_\kappa\left(N,\gamma\right). \tag{3.28}$$

Proof. In equation (3.8), we have seen that if the coefficients c_i sum to one then the residual of the extrapolation is equal to the combination of residuals, so

$$\left\| g\left(\sum_{i=0}^{N}(c_\gamma^*)_i x_i\right) - \sum_{i=0}^{N}(c_\gamma^*)_i x_i \right\| = \left\| \sum_{i=0}^{N}(c_\gamma^*)_i r_i \right\|.$$

By definition of c_γ^*,

$$\left\| \sum_{i=0}^{N}(c_\gamma^*)_i r_i \right\| = \min_{c:\mathbf{1}^T c=1} \|Rc\| \qquad \text{s.t. } \|c\| \leq \frac{\gamma}{\sqrt{1+N}}.$$

Using the Krylov structure of the matrix R, for any c we have

$$\|Rc\| \leq \|r_0\| \left\| \sum_{i=0}^{N}(c)_i G^i \right\| = \|r_0\| \left\| \sum_{i=0}^{N} p(G) \right\|.$$

where p is a polynomial of degree N with coefficient c. We thus have as minimization problem,

$$\left\| \sum_{i=0}^{N}(c_\gamma^*)_i r_i \right\| \leq \min_{p\in\mathbb{R}_{[N]}:p(1)=1} \|r_0\| \left\| \sum_{i=0}^{N} p(G) \right\| \qquad \text{s.t. } \|p\| \leq \frac{\gamma}{\sqrt{1+N}}.$$

Since $0 \preceq G \preceq 1-\kappa$, after maximization over all matrices G with spectrum contained in $[0,(1-\kappa)]$,

$$\left\| \sum_{i=0}^{N}(c_\gamma^*)_i r_i \right\| \leq \|r_0\| \min_{p\in\mathbb{R}_{[N]}:p(1)=1} \max_{x\in[0,(1-\kappa)]} \left\| \sum_{i=0}^{N} p(x) \right\| \qquad \text{s.t. } \|p\| \leq \frac{\gamma}{\sqrt{1+N}}.$$

The right-hand-side is *exactly* the definition of the constrained Chebyshev polynomial, thus

$$\left\| \sum_{i=0}^{N}(c_\gamma^*)_i r_i \right\| \leq \|r_0\| S_\kappa(N,\gamma).$$

■

This result will be useful in Proposition 3.3.6 for bounding the norm $\|\tilde{g}(x_{\text{extr}}) - x_{\text{extr}}\|$.

3.3.4 Convergence rate

We will now prove global accuracy bounds. Let the extrapolation be

$$x_{\text{extr}} = \sum_{i=0}^{N} (\tilde{c}_\gamma^*)_i \tilde{x}_i.$$

where $\tilde{c}_\gamma^*$ is the solution of

$$\min_{c:\mathbf{1}^T c=1} \|\tilde{R}c\|, \quad \|c\| \leq \frac{\gamma}{\sqrt{1+N}}. \tag{3.29}$$

We recall that this way of computing the coefficients is not that different from Algorithm 2. In fact, it suffices to set some λ, compute c_λ^* using Algorithm 2 then set $\gamma = \|c_\lambda^*\|\sqrt{1+N}$.

We will link the norm of the residual of the extrapolation,

$$\|\tilde{g}(x_{\text{extr}}) - x_{\text{extr}}\|,$$

with the optimal value of the optimization problem (3.29). We base the proof the the following decomposition of the residual norm,

$$\begin{aligned}
\|\tilde{g}(x_{\text{extr}}) - x_{\text{extr}}\| &\leq \|g(x_{\text{extr}}) - x_{\text{extr}}\| + \|e_{\text{extr}}\|, \\
&= \left\| g(x_{\text{extr}}) + \sum_{i=0}^{N} (\tilde{c}_\gamma^*)_i e_i - \sum_{i=0}^{N} (\tilde{c}_\gamma^*)_i e_i - x_{\text{extr}} \right\| + \|e_{\text{extr}}\|, \\
&= \left\| \sum_{i=0}^{N} (\tilde{c}_\gamma^*)_i \tilde{x}_{i+1} - \sum_{i=0}^{N} (\tilde{c}_\gamma^*)_i e_i - x_{\text{extr}} \right\| + \|e_{\text{extr}}\|, \\
&\leq \underbrace{\left\| \sum_{i=0}^{N} (\tilde{c}_\gamma^*)_i \tilde{r}_i \right\|}_{\textbf{Acceleration}} + \underbrace{\left\| \sum_{i=0}^{N} (\tilde{c}_\gamma^*)_i e_i \right\|}_{\textbf{Stability}} + \underbrace{\|e_{\text{extr}}\|}_{\textbf{Local Error}},
\end{aligned} \tag{3.30}$$

where e_i is defined in equation (Pert. LFPI) and $e_{\text{extr}} = \tilde{g}(x_{\text{extr}}) - g(x_{\text{extr}})$. The bound will be *local* because we have no control on the perturbation $\|e_{\text{extr}}\|$. The next theorem bounds separately the two first terms using constrained Chebyshev polynomials.

Proposition 3.3.6. *Let x_{extr} be computed using (2), and the matrix $E = [e_0, e_1, \ldots]$ be the matrix of noises. Then, the residual of the extrapolation x_{extr} computed by Algorithm 2 satisfies*

$$\|\tilde{g}(x_{extr}) - x_{extr}\| \leq \|r_0\| S_\kappa(N, \gamma) + \frac{\gamma}{\sqrt{1+N}} \left(\|\tilde{R} - R\| + \|E\| \right) + \|e_{extr}\|. \tag{3.31}$$

Proof. We first bound the term **Stability**. Since we have the constraint $\|c_\gamma\| \leq \frac{\gamma}{\sqrt{1+N}}$,

$$\|\sum_{i=0}^{N}(\tilde{c}_\gamma^*)_i e_i\| \leq \|E\|\frac{\gamma}{\sqrt{1+N}}.$$

Now we bound the term **Acceleration**. By definition of $\tilde{c}_\gamma^*$,

$$\|\tilde{R}\tilde{c}_\gamma^*\| = \min_{c:\mathbf{1}^T c=1} \|\tilde{R}c\| \quad \text{s.t. } \|c\| \leq \frac{\gamma}{\sqrt{1+N}}.$$

In addition,

$$\|\tilde{R}c\| \leq \|Rc\| + \|(\tilde{R}-R)c\|.$$

Again, due to the constraint on $\|c\|$,

$$\|\tilde{R}\tilde{c}_\gamma^*\| \leq \frac{\gamma\|\tilde{R}-R\|}{\sqrt{1+N}} + \min_{c:\mathbf{1}^T c=1} \|Rc\| \quad \text{s.t. } \|c\| \leq \frac{\gamma}{\sqrt{1+N}}.$$

The right-hand-side is bounded in Proposition 3.3.5, thus

$$\|\tilde{R}\tilde{c}_\gamma^*\| \leq \frac{\gamma\|\tilde{R}-R\|}{\sqrt{1+N}} + \|r_0\|S_\kappa(N,\gamma).$$

Finally, putting all together,

$$\|\tilde{g}(x_{\text{extr}}) - x_{\text{extr}}\| \leq \|r_0\|S_\kappa\left(N,\gamma\right) + \frac{\gamma}{\sqrt{1+N}}\left(\|\tilde{R}-R\| + \|E\|\right) + \|e_{\text{extr}}\|.$$

■

In these bound, we clearly see that the constraint on the norm limits the impact of perturbations. The next proposition can get rid of the term $\|e_{\text{extr}}\|$, but at a cost of a factor $\frac{1}{\kappa}$. The idea consists in bounding $\|x_0 - x^*\|$ rather than the residual. This bound better controls the impact of perturbation, because they all are multiplied by γ.

Proposition 3.3.7. *The distance of the extrapolated point x_{extr} to the limit x^* is bounded by*

$$\|x_{extr} - x^*\| \leq \frac{1}{\kappa}\left(\|x_0 - x^*\|S_\kappa\left(N,\gamma\right) + \frac{\gamma}{\sqrt{1+N}}\left(\|\tilde{R}-R\| + \|E\|\right)\right). \tag{3.32}$$

Proof. Indeed,

$$\|x_{\text{extr}} - x^*\| = \|(G-I)^{-1}(G-I)(x_{\text{extr}} - x^*)\| \leq \|(G-I)^{-1}\|\|g(x_{\text{extr}}) - x_{\text{extr}}\|$$

Since $-I \preceq G - I \preceq -\kappa I$, we have

$$\|x_{\text{extr}} - x^*\| \leq \frac{1}{\kappa}\|g(x_{\text{extr}}) - x_{\text{extr}}\|.$$

The term $\|g(x_{\text{extr}}) - x_{\text{extr}}\|$ corresponds to the sum of **Acceleration** and **Stability** in equation (3.30), bounded in Theorem 3.3.6. Finally,

$$\|r_0\| = \|(G - I)(x_0 - x^*)\| \leq \|x_0 - x^*\|.$$

Because $-I \preceq G - I \preceq -\kappa I$, thus $\|G - I\| \leq 1$. Combining the two bounds together leads to the desired result. ■

We will now link $\tilde{R} - R$ with E, if we assume all the perturbation are bounded by a maximum perturbation ϵ.

Proposition 3.3.8. *Assume $\|e_i\| \leq \epsilon \; \forall i = 0 \dots N$. Then,*

$$\frac{1}{\sqrt{1+N}} \left(\|\tilde{R} - R\| + \|E\| \right) \leq \left(1 + \frac{2}{\kappa} \right) \epsilon$$

Proof. We begin by the bound on $\|E\|$. Since $\|e_i\| \leq \epsilon$,

$$\|E\| \leq \sqrt{\sum_{i=0}^{N} \|e_i\|^2} \leq \epsilon \sqrt{N+1}.$$

The error $\|\tilde{R} - R\|$ can be computed column-wise, where

$$\|\tilde{R} - R\| \leq \sum_{i=1}^{N} \|\tilde{r}_i - r_i\| \leq 2 \sum_{i=1}^{N} \|\tilde{x}_i - x_i\|$$

However,

$$\begin{aligned} \tilde{x}_i - x_i &= g(\tilde{x}_{i-1}) + e_{i-1} - g(x_{i-1}), \\ &= g(\tilde{x}_{i-1} - x_{i-1}) + e_{i-1}, \\ &= e_{i-1} + G e_{i-2} + G^2 e_{i-3} \dots = \sum_{j=1}^{i} G^{i-j} e_{j-1}, \end{aligned}$$

whose norm is bounded by

$$\|\tilde{x}_i - x_i\| \leq \sum_{j=1}^{i} (1-\kappa)^{i-j} \|e_{j-1}\| \leq \sum_{k=0}^{i-1} (1-\kappa)^k \epsilon = \frac{1 - (1-\kappa)^{i-1}}{\kappa} \epsilon.$$

To simplify further results, we will use the coarse bound

$$\frac{1 - (1-\kappa)^{i-1}}{\kappa} \epsilon \leq \frac{\epsilon}{\kappa}.$$

This bounds $\|\tilde{R} - R\|$ by

$$2\frac{\sqrt{1+N}}{\kappa}\epsilon$$

The sum of the two bounds leads to the desired result. ■

We now have the final expression of the accuracy bound for the RNA algorithm.

Theorem 3.3.9. *Assume we apply the RNA algorithm on a sequence of vector generated by* (Pert. LFPI)*, where $\|G\| \leq 1-\kappa$ and the norm of the perturbation $\|e_i\|$ is bounded by ϵ. Let the norm of the coefficients $\tilde{c}_\lambda^*$ computed by Algorithm 2 be bounded by $\frac{\gamma}{\sqrt{N+1}}$ with $\gamma \geq 1$. Then, the accuracy $\|x_{extr} - x^*\|$ is bounded by*

$$\|x_{extr} - x^*\| \leq \|x_0 - x^*\| \frac{1}{\kappa}\Bigg(\underbrace{S_\kappa(N,\gamma)}_{\textbf{\textit{Accelerated rate}}} + \underbrace{\frac{\gamma\epsilon}{\|x_0 - x^*\|}\left(1 + \frac{1}{\kappa}\right)}_{\textbf{\textit{Impact of perturbation}}}\Bigg). \tag{3.33}$$

The next proposition gives us a condition on γ to recover an optimal rate of convergence when the perturbation becomes smaller.

Proposition 3.3.10. *Assume we generate a sequence $\tilde{x}_i$ from* (Pert. LFPI)*. If we apply Algorithm 2 with λ s.t. $\|\tilde{c}_\lambda^*\| \leq \frac{\gamma}{\sqrt{1+N}}$, where γ satisfies*

$$\gamma = \gamma(\epsilon), \qquad \gamma \geq 1, \qquad \lim_{\epsilon \to 0} \frac{\gamma}{\sqrt{1+N}} = \infty \qquad \textit{and} \qquad \lim_{\epsilon \to 0} \frac{\gamma\epsilon}{\|x_0 - x^*\|} = 0,$$

then we asymptotically recover the optimal rate (3.9) *with a factor $\frac{1}{\kappa}$, i.e.,*

$$\|x_{extr} - x^*\| \leq \frac{\|x_0 - x^*\|}{\kappa} \frac{2\beta^N}{1+\beta^{2N}}.$$

Proof. The proof is straightforward using Theorem 3.2.2. It suffices to remark that $S_\kappa(N, \infty)$ corresponds to an unconstrained Chebychev polynomial. ■

In short, this bounds tells us that once the perturbations become small, the norm of the coefficients should be large, but not too large in comparison with the ratio between the perturbation and the distance to the optimum $\|x_0 - x^*\|$. This is an expected result, since larger coefficients lead to a better acceleration term, while keeping the norm not too large avoid stability issues.

3.3.5 Regularized Acceleration for Stochastic Algorithms

If we assume the perturbations e_i are sampled from some probability distribution whose first and second moments are bounded, then we also recover an accelerated rate of convergence in expectation.

Proposition 3.3.11. *Assume we generate a sequence* $\{\tilde{x}_i\}$ *from* (Pert. LFPI). *Let* e_i *be a random noise where*

$$\|\mathbb{E}[e_i]\| \leq \nu, \qquad \|\mathbb{E}[e_i - \mathbb{E}[e_i]]\|^2 \leq \sigma^2.$$

Assume $\|\tilde{c}^*_\lambda\| \leq \frac{\gamma}{\sqrt{1+N}}$. *In this case, the bound of Theorem 3.3.9 holds in expectation with* $\epsilon = \nu + \sigma$.

Proof. Indeed, from Theorem 3.3.9,

$$\mathbb{E}\left[\|x_{\text{extr}} - x^*\|\right] \leq \|x_0 - x^*\| \frac{1}{\kappa}\left(S_\kappa(N,\gamma) + \mathbb{E}\left[\frac{\gamma\epsilon}{\|x_0 - x^*\|}\left(1 + \frac{1}{\kappa}\right)\right]\right).$$

In fact, there is no source of noise in $S_\kappa(N,\gamma)$, since it corresponds to the rate of convergence in the noiseless case. In the right hand side, the only source of randomness is ϵ, which bounds

$$\|\tilde{x}_i - x_i\| \leq \|e_i\| \leq \epsilon \ \ \forall i.$$

Let $\nu_i = \mathbb{E}[e_i]$. If we take the expectation,

$$\mathbb{E}[\|e_i\|] = \mathbb{E}\left[\|e_i - \nu_i + \nu_i\|\right] \leq \mathbb{E}\left[\|e_i - \nu_i\|\right] + \|\nu_i\|.$$

By assumption, the right hand side is bounded by ν. The left hand side becomes

$$\mathbb{E}\left[\|e_i - \nu_i\|\right] = \mathbb{E}\left[\sqrt{\|e_i - \nu_i\|^2}\right] \leq \sqrt{\mathbb{E}\left[\|e_i - \nu_i\|^2\right]},$$

which, by assumption, is bounded by σ. ■

In short, this proposition ensures acceleration even in the presence of random perturbation, if the ratio

$$\frac{\nu + \sigma}{\|x_0 - x^*\|}$$

is small. In practice, if we have a noise independent of the iterates, then this is not always true, especially when we get closer to the optimum x^*. Intuitively, this means the magnitude of the noise becomes too large in comparison with the residues. In other words, the "signal to noise" ratio becomes worse as we approach the limit x^*. We will see that this will not be the case with variance-reduced algorithms.

3.4 Computational Complexity of Convergence Acceleration

In Algorithm 2, computing the coefficients $\tilde{c}^*_\lambda$ means solving the $N \times N$ system $(\tilde{R}^T\tilde{R} + \lambda I)z = \mathbf{1}$. We then get $\tilde{c}^*_\lambda = z/(\mathbf{1}^T z)$. This can be done in both batch and online mode. We will see that,

in any case, we end up with a complexity of $O(dN^2 + N^3)$, for a small value of N (usually, $N \in [5, 20]$). The complexity of the acceleration algorithm is linear in the dimension, thus adding a negligible additional computation cost to the original procedure.

3.4.1 Online updates

Here, we receive the vectors $\tilde{r}_i$ one by one from the optimization algorithm, and we would like to solve the linear system in parallel of the optimization algorithm. In this case, we perform low-rank updates on the Cholesky factorization of the system matrix [29]. At iteration i, we have the Cholesky factorization $LL^T = \tilde{R}^T\tilde{R} + \lambda I$, where L is a triangular matrix. We receive a new vector r_+ and we want

$$L_+L_+^T = \begin{bmatrix} L & 0 \\ a^T & b \end{bmatrix} \begin{bmatrix} L^T & a \\ 0 & b \end{bmatrix} = \begin{bmatrix} \tilde{R}^T\tilde{R} + \lambda I & \tilde{R}^T r_+ \\ (\tilde{R}^T r_+)^T & r_+^T r_+ + \lambda \end{bmatrix}.$$

We can explicitly solve this system in variables a and b, and the solutions are

$$a = L^{-1}\tilde{R}^T r_+, \quad b = a^T a + \lambda.$$

The complexity of this update is thus $O(i\,d + i^2)$, i.e. the matrix-vector multiplication of $\tilde{R}^T r_+$ with cost $O(i\,d)$ and solving a $i \times i$ triangular system with cost $O(i^2)$. Since we need to do it N times, the final complexity is thus $O(dN^2 + N^3)$.

3.4.2 Batch mode

The complexity is divided in two parts: First, we need to build the linear system itself. Since $\tilde{R} \in \mathbb{R}^{d\times N}$, it takes $O(dN^2)$ flops to perform the multiplication. Then we need to solve the linear system $(\tilde{R}^T\tilde{R} + \lambda I)z = \mathbf{1}$ which can be done by Gaussian elimination (in particular when N is small), by Cholesky factorization or by using an iterative method like conjugate gradient. It takes $O(N^3)$ flops to solve the linear system in the worst case, meaning that the overall complexity is $O(dN^2 + N^3)$.

3.5 Online Regularized Nonlinear Acceleration and Adaptive Momentum-based Algorithms

So far, we analyzed an algorithm which extrapolates the limit point x^* using the iterates from (Pert. LFPI). However, these results are limited to single-step algorithms such as gradient descent (due to the assumption that the matrix G in (Pert. LFPI) has to be symmetric), thus excluding much faster momentum-based methods such as Nesterov's fast gradient algorithm.

3.5.1 Accelerating a Class of Non-symmetric Algorithms

Here, we will add an additional structural assumption on the dynamic of the points x_i, allowing (variable) linear combinations between two calls of the function g,

$$\begin{cases} x_i &= G(y_{i-1} - x^*) + x^* \qquad (\Leftrightarrow Ax + b), \\ y_i &= \sum_{j=1}^{i} \alpha_j^{(i)} x_j + \sum_{j=0}^{i-1} \beta_j^{(i)} y_j, \end{cases} \tag{Comb. LFPI}$$

and the perturbed version,

$$\begin{cases} \tilde{x}_i &= G(\tilde{y}_{i-1} - x^*) + x^* + e_i, \qquad (\Leftrightarrow \tilde{x}_i = g(y_{i-1}) + \epsilon_i), \\ \tilde{y}_i &= \sum_{j=1}^{i} \alpha_j^{(i)} \tilde{x}_j + \sum_{j=0}^{i-1} \beta_j^{(i)} \tilde{y}_j, \end{cases} \tag{Comb. Pert. LFPI}$$

where the coefficient α and β are arbitrary. We will see in the next chapter that this class of algorithms includes, for example, momentum-based optimization methods. We wrote x^* explicitly in the algorithm to simplify proofs and notations, but in practice this quantity is of course not known, and does not need to be known when using the acceleration method.

It is not possible to apply RNA directly on this sequence since it cannot be written as (Pert. LFPI). This is mainly due to the dependence of x_i to the previous $x_{i-1}, x_{i-2}, \dots, x_1$. Even if we augment artificially the states of the problem, for example with the concatenation

$$z_i = \begin{bmatrix} x_i;\ x_{i-1};\ \dots;\ y_{i-1};\ \dots \end{bmatrix}^T$$

then the underlying matrix G describing the algorithm is not symmetric, so our convergence analysis does not apply. In addition, the coefficients $\alpha_j^{(i)}$ and $\beta_j^{(i)}$ may change between two iterations, thus resulting in a time-varying matrix G_i. Hopefully, it is possible to slightly modify RNA to overcome this difficulty. It suffices to use the residues $r_i = x_{i+1} - y_i$ instead of $x_{i+1} - x_i$ as described in Algorithm 3.

Algorithm 3 Regularized Nonlinear Acceleration (Improved version)

Require: Iterates $\tilde{y}_0, \tilde{y}_1, ..., \tilde{y}_N$; $\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_{N+1}$ from (Comb. Pert. LFPI), regularization parameter $\lambda > 0$.

1: Compute $\tilde{R} = [\tilde{r}_0, ..., \tilde{r}_N]$, where $\tilde{r}_i = \tilde{x}_{i+1} - \tilde{y}_i$

2: Solve

$$\tilde{c}_\lambda^* = \operatorname*{argmin}_{c^T 1 = 1} \|\tilde{R}c\|^2 + \lambda \|c\|^2,$$

or equivalently solve $(\tilde{R}^T \tilde{R} + \lambda I) z = \mathbf{1}$ then set $\tilde{c}_\lambda^* = z / \mathbf{1}^T z$.

Ensure: Approximation of x^* computed as $\sum_{i=1}^{N+1} (\tilde{c}_\lambda^*)_i \tilde{x}_i$

The next Theorem shows that when there are no perturbations, under mild assumptions on the coefficients we recover the optimal rate in Theorem 3.2.2 if we do not use regularization.

Theorem 3.5.1. *Let the sequences of points* $\{y_0 \dots y_{N-1}\}$ *and* $\{x_1 \dots x_N\}$ *be generated by* (Comb. LFPI) *with* $0 \preceq G \preceq (1-\kappa)I$. *Let*

$$\sum_{j=1}^{i} \alpha_j^{(i)} + \sum_{j=0}^{i-1} \beta_j^{(i)} = 1, \qquad \alpha_i^{(i)} \neq 0.$$

If we apply Algorithm 3, the rate of convergence is bounded by

$$\left\| g(x_{extr}) - x_{extr} \right\| \leq \frac{2\beta^N}{1+\beta^{2N}} \left\| r_0 \right\| \qquad \textit{where} \quad \beta = \frac{1-\sqrt{\kappa}}{1+\sqrt{\kappa}} < 1. \tag{3.34}$$

Proof. This proof extend the one of Theorem 3.2.2. We have to show that minimizing the residues

$$\min_{c:\mathbf{1}^T c=1} \left\| \sum_{i=0}^{N} c_i r_i \right\|$$

is equivalent to finding the minimal polynomial

$$\min_{p \in \mathbb{R}_{[N]}: p(1)=1} \|p(G) r_0\|.$$

It is equivalent to prove that r_i can be written as a polynomial in G of degree *exactly* i, whose coefficients sums to one. Indeed, assume we have

$$r_i = p_i(G) r_0, \qquad p_i(1) = 1.$$

In this case, the family of polynomial $\{p_i\}_{i=0\dots N}$ generates the space of polynomials of degree N. We can thus create $p(G)r_0$ for *any* p from a linear combination of r_i. In addition, if the coefficients of this linear combination sum to one, we create a polynomial $p(G)$ whose coefficients sums to one,

$$\sum_{i=0}^{N} c_i p_i(1) = \sum_{i=0}^{N} c_i = 1.$$

Now, we have proved that

$$r_i = p_i(G) r_0, \;\; p(1) = 1 \quad \Rightarrow \quad \min_{c:\mathbf{1}^T c=1} \left\| \sum_{i=0}^{N} c_i r_i \right\| = \min_{p \in \mathbb{R}_N[x]: p(1)=1} \|p(G) r_0\|.$$

The right hand side is already bounded by Theorem 3.2.2. It remains to show the equality, which is true if $r_i = p_i(G)r_0$, where $p(1) = 1$ and $\deg(p_i) = i$. For simpler notations, we remove the superscript $^{(i)}$ for coefficients $\alpha_j^{(i)}$ and $\beta_j^{(i)}$ without loss of generality. We will prove the claim by recursion. We start with the simple case $i = 0$, i.e.,

$$r_0 = I r_0$$

and the identity matrix I indeed corresponds to the polynomial $p(G) = 1 \cdot G^0$ of degree 0, whose coefficients sums to one. Now, assume the claim true for r_{i-1}, and we will show it is also true for r_i. Indeed,

$$\begin{aligned} r_i &= x_{i+1} - y_i, \\ &= G(y_i - x^*) + x^* - y_i, \\ &= (G - I)\left(\sum_{j=1}^{i} \alpha_j x_j + \sum_{j=0}^{i-1} \beta_j y_j - x^*\right). \end{aligned} \tag{3.35}$$

Since $\sum_{j=1}^{i} \alpha_j + \sum_{j=0}^{i-1} \beta_j = 1$,

$$(G - I)\left(\sum_{j=1}^{i} \alpha_j x_j + \sum_{j=0}^{i-1} \beta_j y_j - x^*\right) = (G - I)\left(\sum_{j=1}^{i} \alpha_j (x_j - x^*) + \sum_{j=0}^{i-1} \beta_j (y_j - x^*)\right).$$

In addition,

$$x_j - x^* = G(y_{j-1} - x^*) + x^* - x^* = G(y_{j-1} - x^*).$$

Finally, using equation (3.35),

$$(G - I)(y_j - x^*) = r_j.$$

Bringing it all together leads to

$$\begin{aligned} & (G - I)\left(\sum_{j=1}^{i} \alpha_j x_j + \sum_{j=0}^{i-1} \beta_j y_j - x^*\right), \\ = \; & (G - I)\left(\sum_{j=1}^{i} \alpha_j (x_j - x^*) + \sum_{j=0}^{i-1} \beta_j (y_j - x^*)\right), \\ = \; & (G - I)\left(\sum_{j=0}^{i-1} (\alpha_j G + \beta_j I)(y_j - x^*)\right), \\ = \; & \sum_{j=0}^{i-1} (\alpha_{j+1} G + \beta_j I)(G - I)(y_j - x^*), \\ = \; & \sum_{j=0}^{i-1} (\alpha_{j+1} G + \beta_j I) r_j. \end{aligned}$$

We assumed by recursion that $r_j = p_j(G) r_0$, where p_j is a polynomial of degree *exactly* j,

where $p_j(1) = 1$. The previous line is thus written

$$r_i = \underbrace{\sum_{j=0}^{i-1} \Big(\alpha_{j+1} G p_j(G) + \beta_j p_j(G) \Big)}_{= p_i(G)} r_0.$$

Let us check if $p_i(1) = 1$. Indeed,

$$p_i(1) = \sum_{j=0}^{i-1} \Big(\alpha_{j+1} p_j(1) + \beta_j p_j(1) \Big) = \sum_{j=0}^{i-1} \Big(\alpha_{j+1} p_j(1) + \beta_j p_j(1) \Big) = 1.$$

Now, we check if the degree of the polynomial is indeed equal to i. We have

$$\deg(p_i(x)) \quad = \quad \deg \left(\sum_{j=0}^{i-1} (\alpha_j x + \beta_j) p_j(x) \right).$$

Since by assumption $\deg(p_j(x)) < \deg(p_{i-1}(x))$ for $j < i - 1$,

$$\deg(p_i(x)) = \deg \left((\alpha_i x + \beta_{i-1}) p_{i-1}(x) \right).$$

Because $\alpha_i \neq 0$ by assumption,

$$\deg(p_i(x)) = \deg \left(\alpha_i x p_{i-1}(x) \right) = 1 + \deg(p_{i-1}(x)).$$

Finally, because the degree of p_{i-1} is exactly equal to $i - 1$,

$$\deg(p_i) = 1.$$

We proved by recursion that $\deg(p_i) = i$ and $p_i(1) = 1$, which concludes the proof. ■

The previous result also shows that our method is *adaptive*. Whatever the algorithm, if the iteration $g(x)$ converges, the coefficients α_i and β_i sum to one and we have α_N non-zero, we obtain an optimal rate of convergence. In comparison, in many case coefficients α and β are function of κ and aim to improve the rate of convergence of $\|x_i - x^*\|$. In practice, it may be difficult to estimate κ, and thus the coefficients may not be optimal and this lead to sub-optimal rates.

We can now reproduce the bound of Theorem 3.3.9 using similar tricks.

Proposition 3.5.2. *Assume we apply Algorithm 3 on a sequence of vectors generated by* (Comb. Pert. LFPI)*, where the norm of the perturbation $\|e_i\|$ is bounded by ϵ. Let the norm of the coefficients $\tilde{c}^*_\lambda$ computed by Algorithm 3 be bounded by $\frac{\gamma}{\sqrt{N+1}}$ with $\gamma \geq 1$. Then, the*

accuracy $\|x_{extr} - x^*\|$ *is bounded by*

$$\|x_{extr} - x^*\| \leq \|x_0 - x^*\| \, \frac{1}{\kappa}\left(S_\kappa\left(N, \gamma\right) + \mathit{Constant} \cdot \frac{\gamma\epsilon}{\|x_0 - x^*\|} \right). \tag{3.36}$$

Proof. Indeed,

$$\begin{aligned} x_{\text{extr}} &= \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \tilde{x}_{i+1} \\ &= \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \tilde{g}(\tilde{y}_i) \\ &= \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \left(g(\tilde{y}_i) + e_i \right) \end{aligned}$$

Thus, the distance to the optimum is bounded by

$$\|x_{\text{extr}} - x^*\| \leq \left\| \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \left(g(\tilde{y}_i) - x^* \right) \; + \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i e_i \right\|.$$

Because $g(y) - x^* = G(y - x^*)$,

$$\left\| \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \left(g(\tilde{y}_i) - x^* \right) \; + \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i e_i \right\| = \left\| G \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \left(\tilde{y}_i - x^* \right) \; + \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i e_i \right\|.$$

In addition, because $(G - I)(y - x^*) = g(y) - y$,

$$\left\| G \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \left(\tilde{y}_i \right) - x^* \right) \; + \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i e_i \right\| \leq \|(G-I)^{-1}\| \left\| G \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \left(g(\tilde{y}_i) - \tilde{y}_i \right) \; + (G - I) \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i e_i \right\|.$$

Finally, because $g(\tilde{y}_i) + e_i = \tilde{r}_i$, $\|(G - I)^{1-}\| \leq 1/\kappa$ and $\|G\| \leq 1 - \kappa$,

$$\|(G-I)^{-1}\| \left\| G \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \left(g(\tilde{y}_i) - \tilde{y}_i \right) \; + (G - I) \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i e_i \right\| \leq \frac{1 - \kappa}{\kappa} \left(\left\| \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i \tilde{r}_i \right\| + \left\| \sum_{i=0}^{N} (\tilde{c}_\lambda^*)_i e_i \right\| \right).$$

Here, we recover exactly the structure in equation (3.30) (without the term $\|e_{\text{extr}}\|$), which was the basis of the proof in Theorem 3.3.9. ■

Theses results allows us to introduce the *online-RNA* algorithm, which consists in using RNA at *every* iterations.

3.5.2 Online RNA

In the RNA algorithm, the output corresponds to a linear combination of the points x_i, whose coefficients sums to one. It corresponds, in fact, to what we described in equation (Comb. Pert. LFPI) where, at iteration N,

$$\alpha^{(N)} = \tilde{c}_\lambda^*, \qquad \beta^{(N)} = 0.$$

This describes the Online RNA algorithm.

Algorithm 4 Online Regularized Nonlinear Acceleration

Require: Function $g(x)$ with fixed-point x^*, starting point $\tilde{y}_0$, regularization λ.

1: $i = 1$.
2: **while** the norm of residual $\|g(\tilde{y}_{i-1}) - \tilde{y}_{i-1}\| \geq$ Tolerance **do**
3: Compute the next iterate $\tilde{x}_i = g(\tilde{y}_{i-1})$.
4: Compute the extrapolation $\tilde{y}_i = \mathbf{RNA}(\{\tilde{x}_j\}, \{\tilde{y}_j\}, \lambda)$ (See Algorithm 3).
5: $i = i + 1$.
6: **end while**

Since this algorithm matches the description of (Comb. Pert. LFPI), it satisfies the assumptions of Theorem 3.5.1, and thus presents an optimal rate of convergence on linear fixed-points iterations. Contrary to many acceleration techniques, this can be done *without any knowledge* on the problem's constants.

Since the complexity grows quadratically with N (the number of points in the sequences $\{\tilde{x}_i\}$ and $\{y_i\}$), Algorithm 4 is not convenient because because N continuously increases over time. In practice, it is better to use a fixed window where N is bounded. This strategy is similar to the limited version of BFGS, where a fixed number of gradients is stored in memory [11].

3.5.3 Adaptive Momentum-based Algorithms

Previously, we showed two different aspects of the acceleration of scheme (Comb. Pert. LFPI). One is capable to extrapolates x_{extr}, an approximation of the solution x^*, from sequences $\{\tilde{x}_i\}$ and $\{\tilde{y}_i\}$. The other takes as input an iterative method g and accelerate it using RNA at each step. Ideally, we would like to combine the two approaches: using RNA at each iteration on algorithm (Comb. Pert. LFPI). In this section, we will try to unify these two strategies.

Assume that scheme (Comb. Pert. LFPI) has a proven convergence rate, which bounds the error at each iteration. This rate at iteration N is ensured if the condition

$$h_N(\tilde{x}_N, \tilde{y}_{N-1}) \leq 0$$

is satisfied, for some function h_N. For example, in the context of optimization, this may correspond to a sufficient descent condition for the point $\tilde{x}_N$ and $g(x)$ is a simple gradient step with

a certain step length. Assume this condition is automatically satisfied if

$$\tilde{x}_N = g(\tilde{y}_{N-1}).$$

Again, in optimization, this is true when the step length is equal to $1/L$. We will describe a way to combine the RNA algorithm without losing the theoretical guarantee of the original algorithm. For simplicity, we consider $\beta_i = 0$ but it is possible to extend to the case where β is non-zero.

Ideally, we would like to use acceleration as much as we can, but without losing the potential convergence guarantees of the algorithm. We will see in the next proposition that is some cases, it is possible to test a condition which decides if we should use acceleration or not. This test is based on the application of the function h to a "conditional point" z_i.

Proposition 3.5.3. *Assume a convergence rate is proved for sequences $\{\tilde{x}_i\}$ and $\{\tilde{y}_i\}$ if*

$$\begin{aligned} &h_N(\tilde{x}_N, \tilde{y}_{N-1}) \leq 0, \\ &y_N = \sum_{i=1}^{N} \alpha_i \tilde{x}_i, \qquad \alpha_N \neq 0, \quad \sum_{i=1}^{N} \alpha_i = 1 \end{aligned}$$

*where the first condition is ensured when $\tilde{x}_N = g(\tilde{y}_{N-1})$ (we omitted the superscript $^{(N)}$ for clarity). Let $\tilde{c}^*_\lambda$ be the coefficients computed by **RNA** with sequences $\{\tilde{x}_i\}$ and $\{\tilde{y}_i\}$. Consider the "conditional point"*

$$\tilde{z}_N = \sum_{j=1}^{N} \rho_i \tilde{x}_i \qquad \textit{where} \qquad \rho_N = \frac{(\tilde{c}^*_\lambda)_N}{\alpha_N}, \quad \rho_{j<N} = \frac{(\tilde{c}^*_\lambda)_j - \alpha_j}{\alpha_N}.$$

Then, the rate of convergence is preserved if

$$\begin{aligned} &h_N(\tilde{z}_N, \tilde{y}_{N-1}) \leq 0, \\ &y_N = \sum_{j=1}^{N} (\tilde{c}^*_\lambda)_j \tilde{x}_j. \end{aligned}$$

This means we can use acceleration if $h_N(\tilde{z}_N, \tilde{y}_{N-1}) \leq 0$. Otherwise, we should perform a step of the original algorithm.

Proof. Since the convergence rate is proved for $\{\tilde{x}_i\}$ and $\{\tilde{y}_i\}$, it is also proved for any z, $\tilde{y}_{N-1}$ s.t.

$$h(z, \tilde{y}_{N-1}) \leq 0, \qquad y_N = \alpha_N z + \sum_{i=1}^{N-1} \alpha_i \tilde{x}_i.$$

In particular, this is true for $z = \tilde{z}_N$. In this case, $\tilde{y}_N$ becomes

$$\tilde{y}_N = \alpha_N \tilde{z}_N + \sum_{i=1}^{N-1} \alpha_i \tilde{x}_i.$$

If we replace $\tilde{z}_N$ by its expression,

$$\begin{aligned}\tilde{y}_N &= \alpha_N \sum_{i=1}^{N} \rho_i \tilde{x}_i + \sum_{i=1}^{N-1} \alpha_i \tilde{x}_i, \\ &= (\tilde{c}_\lambda^*)_N \tilde{x}_N \sum_{i=1}^{N-1} (\alpha_i + (\tilde{c}_\lambda^*)_j - \alpha_j)\tilde{x}_i, \\ &= \sum_{i=1}^{N} (\tilde{c}_\lambda^*)_i \tilde{x}_i.\end{aligned}$$

■

This theorem shows that the conditional point $\tilde{z}_N$ is crucial the design of an adaptive momentum-based algorithm, and determines if we have to use the adaptive coefficient $\tilde{c}_\lambda^*$ or the user defined coefficients α and β.

Since the coefficients $\tilde{c}_\lambda^*$ are adaptive, we can usually expect a better rate of convergence when using the RNA algorithm. However, we have seen in Theorem 3.3.9 that the upper bound for the accuracy is local and may be very bad in some situation.

Combining the two situations, i.e., the adaptivity of RNA and the convergence guarantees of algorithm (Comb. Pert. LFPI) takes the best of the two worlds, at the cost of one evaluation of $h(\cdot, \tilde{y}_{N-1})$. We formalize the Online RNA applied to momentum-based methods in Algorithm 5.

Algorithm 5 Online Regularized Nonlinear Acceleration for Momentum-Based Methods

Require: Algorithm (Comb. Pert. LFPI), condition function $h(\cdot, \cdot)$, starting point $\tilde{y}_0$, regularization λ.

$i = 1$.
while the norm of residual $\|g(\tilde{y}_{i-1}) - \tilde{y}_{i-1}\| \geq$ Tolerance **do**
  Compute the next iterate $\tilde{x}_i = g(\tilde{y}_{i-1})$.
  Compute the extrapolation $\left[x_{\text{extr}};\ \tilde{c}_\lambda^*\right] = \mathbf{RNA}\big(\{\tilde{x}_j\}, \{\tilde{y}_j\}, \lambda\big)$.
  Compute the conditional point $\tilde{z}_i = \left((\tilde{c}_\lambda^*)_i \tilde{x}_N + \sum_{j=0}^{i-1} \left((\tilde{c}_\lambda^*)_j - \alpha_j\right) x_j^{(i)}\right) \left(1/\alpha_i^{(i)}\right)$.
  if the condition $h(\tilde{z}_i, \tilde{y}_{i-1}) \leq 0$ is satisfied **then**
    Use the extrapolation $\tilde{y}_i = x_{\text{extr}}$.
  else
    Use the original combination $\tilde{y}_i = \sum_{j=1}^{i} \alpha_j^{(i)} \tilde{x}_j$.
  end if
  $i = i + 1$.
end while

This algorithm, capable to combine robustness, adaptive acceleration and theoretical guarantees, concludes this chapter on generic acceleration.

3.6 Conclusion and Perspectives

In this chapter, we introduced the Regularized Nonlinear Acceleration technique, which is able to accelerate the perturbed fixed-point operation with linear combination described in equation (Comb. Pert. LFPI), if the underlying matrix G is symmetric. We deduced a rate of convergence, whose expression depends on the maximum value of a constrained Chebyshev Polynomial (3.24) and the ratio between the (maximum) noise level ϵ and the distance to the optimum x^*. Its computational complexity is small in comparison with the computation of $g(x)$, since it is linear in the dimension d when the length of the sequence to extrapolate is small.

This chapter raises two main questions. We have seen that we can accelerate a fixed-point iteration when G is symmetric, but the case where G is not symmetric anymore is less clear, because we have to compute minimal polynomial on the complex plane instead of the real line (due to the presence of complex eigenvalues). In addition, working with non symmetric matrices makes the spectral radius and the norm two different quantities, which makes the analysis even more difficult.

The second question is about the constrained Chebyshev polynomial. Even if we have rates of convergence, computing a constrained Chebyshev polynomial is hard because it involves modeling a sum-of-squares problem, which is then lifted to an SDP formulation. This is known to be not scalable for large dimensions and solving such problem is computationally hard. In addition, we do not have access to an explicit formula, thus preventing further theoretical analysis. Working on a good approximation of constrained Chebyshev polynomial may lead to better understanding of nonlinear acceleration algorithms.

Chapter 4

Nonlinear Acceleration of Optimization Algorithms

Abstract

In this chapter we will apply the generic acceleration technique RNA to several optimization methods in different settings, such as deterministic or stochastic optimization. In the deterministic setting, we will explicitly write the convergence bounds using standard assumption on the objective function, such as smoothness and strong convexity. For the stochastic one, we will analyze the accuracy bounds in function of the noise level and deduce that variance reduction techniques are crucial for acceleration. We provide numerical experiments in both cases. In addition, we use it to train deep neural networks on CIFAR10 and ImageNet data sets, and empirically observe that a straightforward application of RNA makes networks more accurate.

Main references: The results in this chapter have been partially published in [64], [65] (NIPS 2016 and 2017), [67] (ICLR workshop 2018) and [68] (under review NIPS 2018). A revised version of [64] is under review for the journal Mathematical Programming.

4.1 Introduction

In this chapter, we are interested in minimizing a smooth convex function $f(x)$,

$$\min_{x \in \mathbb{R}^d} f(x),$$

in the variable $x \in \mathbb{R}^n$. To do so, we will use a first-order algorithm which can be written under the form of (Comb. Pert. LFPI) (we will drop the tilde $\tilde{}$ notation in this chapter for clarity).

For example, the gradient method, written,

$$x_{i+1} = x_i - h\nabla f(x_i)$$

with step length $h > 0$ matches exactly the definition, where $g(x) = x - h\nabla f(x)$, $\alpha_i^{(i)} = 1$ and $\beta_i = 0$. Here, we will study the impact of RNA on optimization methods for accelerating their convergence.

Since the publication of Nesterov's optimal first-order smooth convex minimization algorithm [52], significant efforts have been focused on either providing more interpretable views on current acceleration techniques, or on replicating these complexity gains using different, more intuitive schemes. Early efforts sought to directly extend the original acceleration result in [52] to broader function classes [50], allow for generic metrics, line searches, produce simpler proofs [7, 55] or adaptive accelerated algorithms [56], etc. More recently however, several authors [20, 43] have started using classical results from control theory to obtain numerical bounds on convergence rates that match the optimal rates. Others have studied the second order ODEs obtained as the limit for small step sizes of classical accelerated schemes, to better understand their convergence [74, 78]. Finally, recent results have also shown how to wrap classical algorithms in an outer optimization loop, to accelerate convergence and reach optimal complexity bounds [45] on certain structured problems.

The approach we propose here is completely different as our method is built on top of existing optimization algorithms. Classical algorithms typically retain only the last iterate or the average [60] of iterates as their best estimate of the optimum, throwing away all the information contained in the converging sequence of iterates. As it is highly wasteful from a statistical perspective, extrapolation schemes estimate the optimum of an optimization problem using a weighted average of the last iterates produced by an algorithm, where the weights depend on the iterates. Overall, nonlinear acceleration has marginal computational complexity. On convex problems, the online version of RNA is competitive with L-BFGS in our experiments and is robust to misspecified strong convexity parameters.

In the case where we can only access a stochastic gradient, we have seen in the previous chapter that all results on acceleration still hold. Such setting is typical in machine learning, for example when f is a sum of convex functions, and we only have access to the gradient of one randomly selected function [39]. Stochastic optimization is typically challenging as classical algorithms are not convergent (for example, gradient descent or Nesterov's accelerated gradient). Even the averaged version of stochastic gradient descent with constant step size does not converge to the solution but to another point whose proximity to the real minimizer depends of the step size [49, 48].

In this chapter, we will explore optimization algorithms which can be written as

$$\begin{aligned} x_i &= g(x_{i-1}) = G(y_{i-1} - x^*) + x^* + e_i, \\ y_i &= \sum_{j=1}^{i} \alpha_j^{(i)} x_j + \sum_{j=0}^{i} \beta_j^{(i)} y_j, \end{aligned} \tag{4.1}$$

where

$$0 \preceq G \preceq (1-\kappa)I, \qquad 0 < \kappa < 1, \qquad \sum_{j=1}^{i} \alpha_j^{(i)} + \sum_{j=0}^{i} \beta_j^{(i)} = 1, \qquad \alpha_j^{(i)} \neq 0.$$

(We recall we dropped the tilde ˜ notation). Then, we will study the impact of acceleration Algorithm 3 and 5 on these optimization methods. We begin with deterministic scheme, then move to stochastic one and finish with the optimization of neural networks.

4.2 Acceleration of Deterministic Algorithms

4.2.1 Introduction

In this section, we consider the minimization of a smooth and strongly convex function f with constants $0 < \mu \leq L$, i.e.,

$$\mu\|y - x\| \leq \|\nabla f(x)\| \leq L\|y - x\| \qquad \forall\, x,\, y \in \mathbb{R}^d$$

where $\|\cdot\|$ stands for the ℓ_2 norm. We also assume the function two times differentiable, so

$$\mu I \preceq \nabla^2 f(x) \preceq LI\,, \qquad \forall x \in \mathbb{R}^d.$$

In addition, we also assume the Hessian is Lipchitz-continuous with constant M, which means

$$\|\nabla^2 f(y) - \nabla^2 f(x)\| \leq M\|y - x\|.$$

This last assumption is necessary to easily bound the error e_i in (4.1). Before going further, we will show that we can reduce any convex function f to a strongly convex one, whose constant depends on the desired accuracy.

4.2.2 Reduction to Strongly Convex Functions

Using a simple regularization trick which we trace back at least to [35], we can extend our results to smooth functions that are not strongly convex. Suppose we seek to solve

$$\min_{x \in \mathbb{R}^n} f(x)$$

in the variable $x \in \mathbb{R}^n$, where $f(x)$ has a Lipschitz continuous gradient with parameter L with respect to the Euclidean norm, but is not strongly convex. Assume for simplicity that the initial iterate x_0 is close enough to the optimum so that $D \geq \|x_k - x^*\|$ for any $k \geq 0$. We can approximate the above problem by

$$\min_{x \in \mathbb{R}^n} f_\varepsilon(x) \triangleq f(x) + \frac{\varepsilon}{2D^2}\|x_0 - x\|_2^2 \tag{4.2}$$

in the variable $x \in \mathbb{R}^n$, where $f_\varepsilon(x)$ has a Lipschitz continuous gradient with parameter $L+\varepsilon/D^2$ with respect to the Euclidean norm, is strongly convex with parameter ε/D^2 with respect to the same norm. Furthermore $f_\varepsilon(x)$ is an ε approximation of $f(x)$ near the optimum and we get

$$\begin{aligned} f(x_k) - f(x^*) &= f_\varepsilon(x_k) - \frac{\varepsilon}{2D^2}\|x_0 - x_k\|_2^2 - f_\varepsilon(x^*) + \frac{\varepsilon}{2D^2}\|x_0 - x^*\|_2^2, \\ &\leq f_\varepsilon(x_k) - f_\varepsilon(x^*) + \frac{\varepsilon}{2}, \\ &\leq f_\varepsilon(x_k) - f_\varepsilon(x_\varepsilon^*) + \frac{\varepsilon}{2}, \end{aligned}$$

using the smoothness of $f_\varepsilon(x)$ and writing x_ε^* the optimum of problem (4.2). It suffices to optimize f_ε up to $\varepsilon/2$ to find an ε-solution for the original problem. The linear convergence of gradient [55] algorithms guarantees

$$f_\varepsilon(x_k) - f_\varepsilon(x_\varepsilon^*) = \frac{(L+\varepsilon)D^2}{2} r^k\,, \qquad r = 1 - \frac{2\varepsilon}{LD^2 + 2\varepsilon}.$$

The number of iterations required to reach a target precision $\varepsilon/2$ is thus bounded by

$$k = O\left(\frac{\log((L+\varepsilon)D^2/\varepsilon)}{\log(1/r)}\right).$$

By replacing the value of r, we have

$$\log(1/r) \sim 1 - r = \frac{LD^2}{\varepsilon},$$

while accelerated algorithms have $r = 1 - \sqrt{\varepsilon/(LD^2+\varepsilon)}$ which yields

$$\log(1/r) \sim \sqrt{\frac{LD^2}{\varepsilon}}.$$

Up to a logarithmic constant, these upper bounds match the complexity of gradient and accelerated gradient methods. Overall, an algorithm for strongly convex function used with this regularization trick recovers an ε-approximated solution. This means we can always reduce a not strongly convex problem to strongly convex one. Of course, in practice one does not wants to use this king of "hack", but strong convexity, for now, is crucial in the analysis of generic acceleration

methods. The extension to convex functions still remains an open problem.

4.2.3 Rate of Convergence

Local Rate of Convergence

We will study several classical optimization algorithm and show they can be written into the form (4.1), but before we will analyze the potential gain from the RNA algorithm. In the case of deterministic optimization, all the algorithms we will list have the same nonlinear function g for generating x_i, which is the simple gradient step

$$g(x) = x - \frac{1}{L}\nabla f(x). \tag{4.3}$$

If we linearize the algorithm around x^*, we get

$$g(x) = x^* + \left(I - \frac{1}{L}\nabla^2 f(x^*)\right)(x - x^*) + O\left(\|x - x^*\|^2\right).$$

We identify here the matrix $G = \left(I - \frac{1}{L}\nabla^2 f(x^*)\right)$, whose eigenvalues lies in $[0, 1-\kappa]$ where $\kappa = \frac{\mu}{L}$ is *the inverse of* the condition number. We will now show that the error is bounded in function of $\|x_0 - x^*\|$ if the hessian is Lipchitz continuous.

Proposition 4.2.1. *(Nesterov and Polyak [57]) Let f be two times differentiable, whose Hessian is Lipchitz-continuous with constant M. Then,*

$$\left\|g(x) - x^* - \left(I - \frac{1}{L}\nabla^2 f(x^*)\right)(x - x^*)\right\| = \|e(x)\| \leq \frac{M}{2L}\|x - x^*\|^2. \tag{4.4}$$

Proof. Indeed,

$$\begin{aligned}
\left\|g(x) - x^* - \left(I - \frac{1}{L}\nabla^2 f(x^*)\right)(x - x^*)\right\| &= \left\|x - \frac{1}{L}\nabla f(x) - x^* - \left(I - \frac{1}{L}\nabla^2 f(x^*)\right)(x - x^*)\right\|, \\
&= \left\|\frac{1}{L}\nabla f(x) - \frac{1}{L}\nabla^2 f(x^*)(x - x^*)\right\|, \\
&= \frac{1}{L}\left\|\nabla f(x) - \nabla f(x^*) - \nabla^2 f(x^*)(x - x^*)\right\|,
\end{aligned}$$

where $\nabla f(x^*) = 0$ since x^* is a critical point. Because

$$\nabla f(x) - \nabla f(x^*) = \int_0^1 \left(\nabla^2 f(x + \tau(x^* - x))\right)(x - x^*)\,\mathrm{d}\tau,$$

we can bound the norm by

$$\begin{aligned}\frac{1}{L}\left\|\nabla f(x)-\nabla f(x^*)-\nabla^2 f(x^*)(x-x^*)\right\| &\leq \frac{1}{L}\left\|\int_0^1\left(\nabla^2 f(x+\tau(x^*-x))-\nabla f(x^*)\right)(x-x^*)\,\mathrm{d}\tau\right\|,\\ &\leq \frac{M}{L}\|x-x^*\|\int_0^1 \tau\,\mathrm{d}\tau,\end{aligned}$$

which is the desired result. ■

If we assume that all the points x_i are going to be inside some region around x^*,

$$\forall i,\quad \|x_i-x^*\|\leq D,$$

then the bound of Theorem 3.3.9 is written

$$\|x_{\text{extr}}-x^*\|\leq D\,\frac{1}{\kappa}\left(S_\kappa\left(N,\gamma\right)+\gamma\frac{2MD}{L}\left(1+\frac{1}{\kappa}\right)\right). \tag{4.5}$$

We can now derive an upper bound for γ, and thus β, to recover the asymptotic rate of convergence in Proposition 3.3.10. It suffices to set γ in function of D such that

$$\lim_{D\to 0}\gamma=\infty,\qquad \lim_{D\to 0}D\gamma=0.$$

The next proposition gives the relation between D and the value of γ (and λ) to satisfy the above condition.

Proposition 4.2.2. *If we have, for* $0<a<1$*,*

$$\gamma=1+O\left(D^{-a}\right)\qquad or\qquad \lambda=O\left(D^{2(1+a)}\right),$$

then,

$$\lim_{D\to 0}\gamma=\infty\qquad and\qquad \lim_{D\to 0}D\gamma=0.$$

This means RNA recovers the rate of convergence of Proposition 3.3.10.

Proof. Indeed, for $0<a<1$, we have

$$\lim_{D\to 0}D^{-a}=\infty\qquad \text{and}\qquad \lim_{D\to 0}D^{1-a}=0.$$

We have to satisfy the following relationship between λ and γ (Proposition 3.3.4)

$$\frac{\|\tilde{R}\|^2}{(\gamma^2-1)}\leq\lambda$$

to ensure $\|c_\lambda^*\|<\frac{\gamma}{\sqrt{1+N}}$. Since $\|\tilde{R}\|^2=O(D^2)$ and $(\gamma^2-1)\geq(\gamma-1)^2$ for $\gamma>1$, then the

following value of λ satisfies the conditions,

$$\lambda = \frac{O(D^2)}{(\gamma - 1)^2}.$$

Replacing γ by $1 + O(D^{-a})$ leads to the desired result. ■

This result is more permissive that in our associated publications [64] and [65], where the range of λ was $D^2 < \lambda < D^{\frac{8}{3}}$, meanwhile we can now go up to D^4.

Global Rate of Convergence

However, in the theoretical bounds, the interactions between the perturbation and the algorithm are not taken into account, because we only study it from a worst-case point of view. However, we can show that there exists λ for which we can ensure global convergence, if $\|\tilde{x}_i - x^*\|$ is converging monotonically. In fact, it suffices to take $\lambda = \infty$ (or equivalently, $\gamma = 1$) to see that

$$\tilde{c}^*_\infty = \frac{1}{N}[1,\ 1,\ \ldots, 1]^T.$$

This corresponds to take the *average* of previous points. If the distance to the optimum converges at rate $(1 - r)^i$, the rate of convergence for the averaged of N point becomes

$$\|(1/N)\sum_{i=0}^{N} \tilde{x}_i - x^*\| \leq (1/N)\sum_{i=0}^{N} \|\tilde{x}_i - x^*\| \leq (1/N)\sum_{i=0}^{N} (1 - r)^i \|x_0 - x^*\| \leq \frac{1}{N\kappa}\|x_0 - x^*\|.$$

Indeed, this slows down the rate of the original algorithm, but one can simply perform a comparison between the extrapolation step and the vanilla step and take the best between the two.

4.2.4 First Order Optimization Algorithms

We will now list several standard optimization algorithms and write them into the form of (Comb. Pert. LFPI). In this section, $g(x)$ corresponds to a gradient step with step-size $h = 1/L$, as described in equation (4.3). This makes the expression of the residual very simple, since it corresponds to the gradient, i.e., $r_i = \frac{1}{L}\nabla f(x_i)$.

Gradient Method with Line-search

As we just described, the gradient method can be written

$$x_i = g(y_{i-1}), \qquad y_i = x_i.$$

However, we often equip it with a (backtracking) line-search to improve its performance. This means we have a variable step-size h_i, and the method is written

$$x_i = x_{i-1} - h_i \nabla f(x_{i-1}).$$

This can be captured by coefficients $\alpha^{(i)}$ and $\beta^{(i)}$ as following,

$$\begin{aligned} x_i &= g(y_{i-1}) \\ y_i &= \underbrace{Lh_i}_{=\alpha_i^{(i)}} x_i + \underbrace{(1 - Lh_i)}_{=\beta_i^{(i)}} y_{i-1} \end{aligned}$$

In this case, we can verify that $y_i = y_{i-1} - h\nabla f(y_{i-1})$, $\alpha_i^{(i)} + \beta_i^{(i)} = 1$ and $\alpha_i^{(i)} \neq 0$.

Fast Gradient Method

We now write the Nesterov's method [55] into the form of (Comb. Pert. LFPI),

$$\begin{aligned} x_i &= g(y_{i-1}) \\ y_i &= \underbrace{(1 + \theta_i)}_{=\alpha_i^{(i)}} x_i \ \underbrace{-\theta_i}_{=\alpha_{i-1}^{(i)}} x_{i-1} \end{aligned}$$

where $\theta_k = \frac{1-\sqrt{\kappa}}{1+\sqrt{\kappa}}$ if we know the strong convexity constant μ. In this case, the algorithm converges with rate $(1 - \sqrt{\kappa})^i$. Otherwise, we can use $\theta_i = \frac{i}{1+i}$, which has a rate of convergence of the order of $\frac{1}{1+i^2}$. If we want to use the adaptive momentum Algorithm 5, the temporary quantity z_i is written

$$z_i = \frac{1}{1 + \theta_i} \left(x_{\text{extr}} + \theta_i x_{i-1} \right).$$

Then, it suffices to check if

$$h(z_i, y_{i-1}) = f(z_i) - f(y_{i-1}) + \frac{1}{2L} \|\nabla f(y_{i-1})\|^2 \leq 0.$$

In that way, we preserve the original global rate of Nesterov's method.

4.3 Numerical Experiments (Deterministic)

We will compare different methods for the optimization of a logistic loss on three datasets, named Sonar [31], Madelon [34] and sido0 [33]. The logistic loss is defined as follows. Let a dataset with feature matrix $\Theta \in \mathbb{R}^{m \times d}$ which contains m vector of d features each. Consider the $\{-1, 1\}$ vector ϑ which assigns each vectors of Θ to a positive or negative label. Then the logistic loss

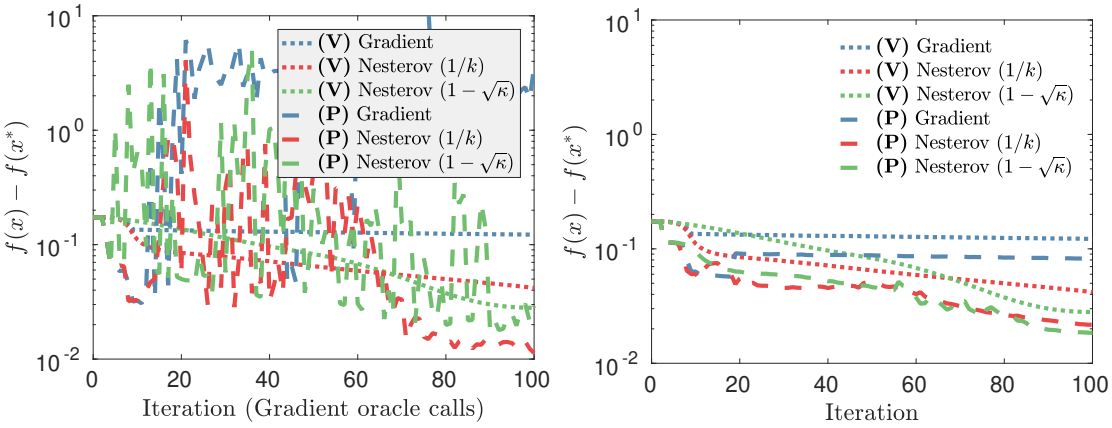


Figure 4.1: Comparison between non-regularized RNA (or Anderson acceleration) and regularized nonlinear acceleration algorithm in the offline fashion (*Left:* $\lambda = 0$ *and Right:* $\lambda = 10^{-8}\|\tilde{R}^T W \tilde{R}\|$). The window size (maximum sequence length) is set to 10. We run the experiment on Madelon dataset, whole regularization is set such that the condition number $\kappa = 10^{-6}$. We clearly see that regularization is crucial for the stability of the algorithm. **(V)** (vanilla) corresponds to the original methods, and **P** (parallel) corresponds to the application of RNA without interaction with the original algorithm.

$\mathcal{L}(w)$ with ℓ_2 penalty τ is defined as

$$\mathcal{L}(w) = \sum_{i=1}^{m} \log(1 + \exp(-\vartheta_i \Theta_i^T w)) + \frac{\tau}{2}\|w\|^2.$$

Its smoothness constant is bounded above by $\|\Theta\|_2^2/4 + \tau$ and its strong convexity parameter is bounded below by τ.

We will test the RNA algorithm on previous methods. We compare three strategies:

- **(V)** Vanilla: we run the original algorithm with no acceleration.
- **(P)** Offline version, in parallel: we run the RNA algorithm on the side (Algorithm 3), with no interaction on the original optimization method.
- **(O)** Online: we use the online version of RNA (Algorithm 5).

4.3.1 Comparison between RNA and Anderson Acceleration

Here, we compare the RNA algorithm with memory size $N = 10$ and regularization parameter $\lambda = 10^{-8}\|\tilde{R}^T \tilde{R}\|$ (where $\tilde{R}$ corresponds to the matrix of gradients $\nabla f(x_i)$ when $g(x)$ is a gradient step) and the *Anderson Acceleration*, which is basically RNA without regularization. In Figure 4.1, we clearly see that the regularization in crucial for acceleration because it plays an important role in the algorithm stability.

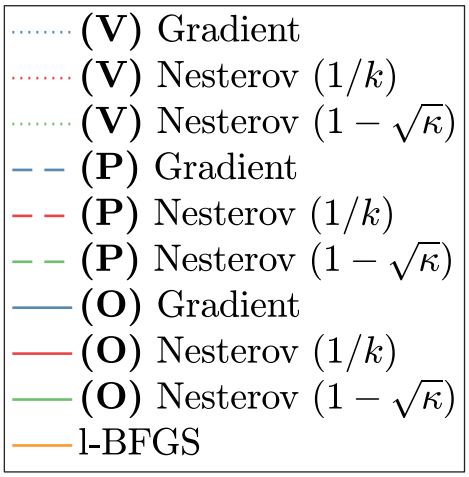


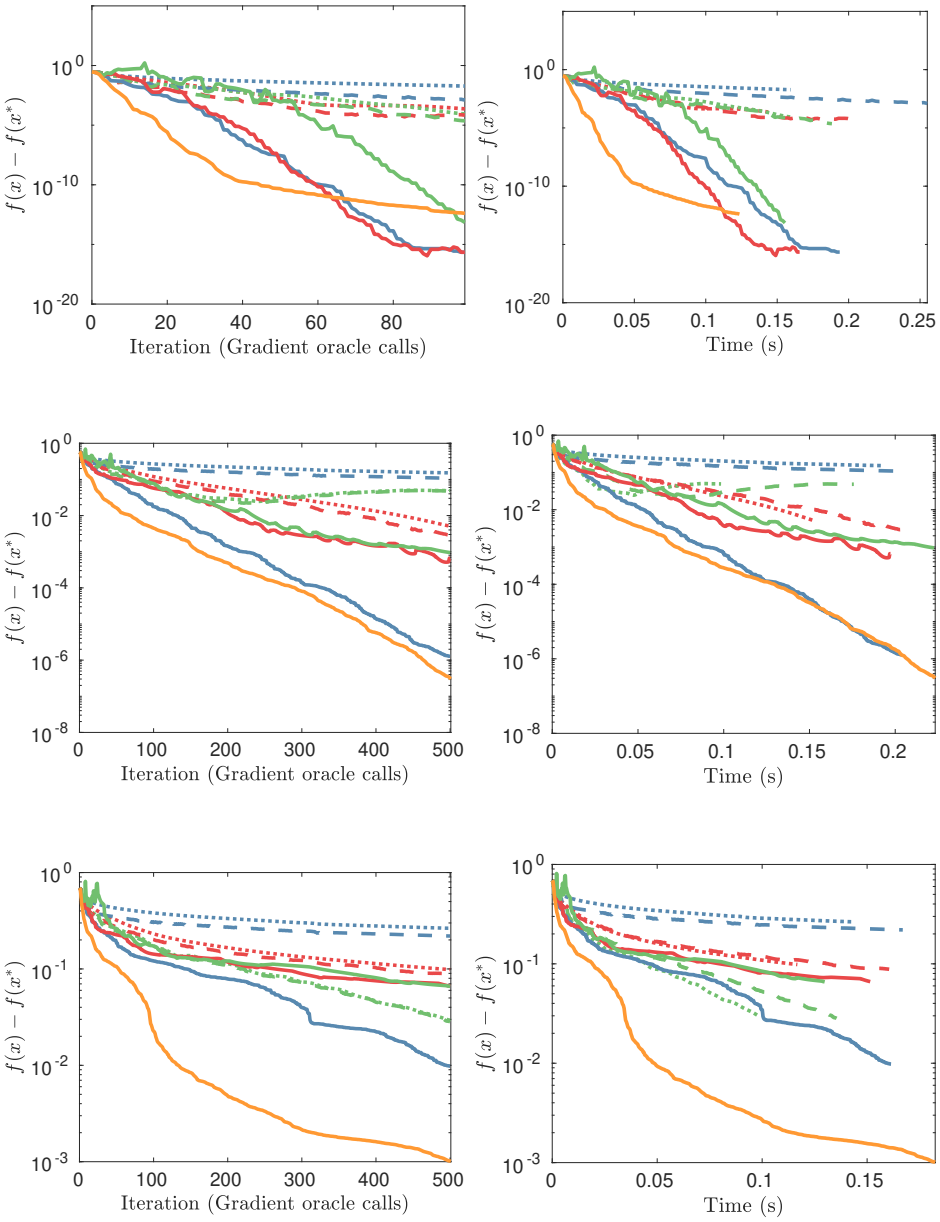
Figure 4.2: Legend for numerical experiments on deterministic algorithms

4.3.2 Comparison Between Acceleration Strategies

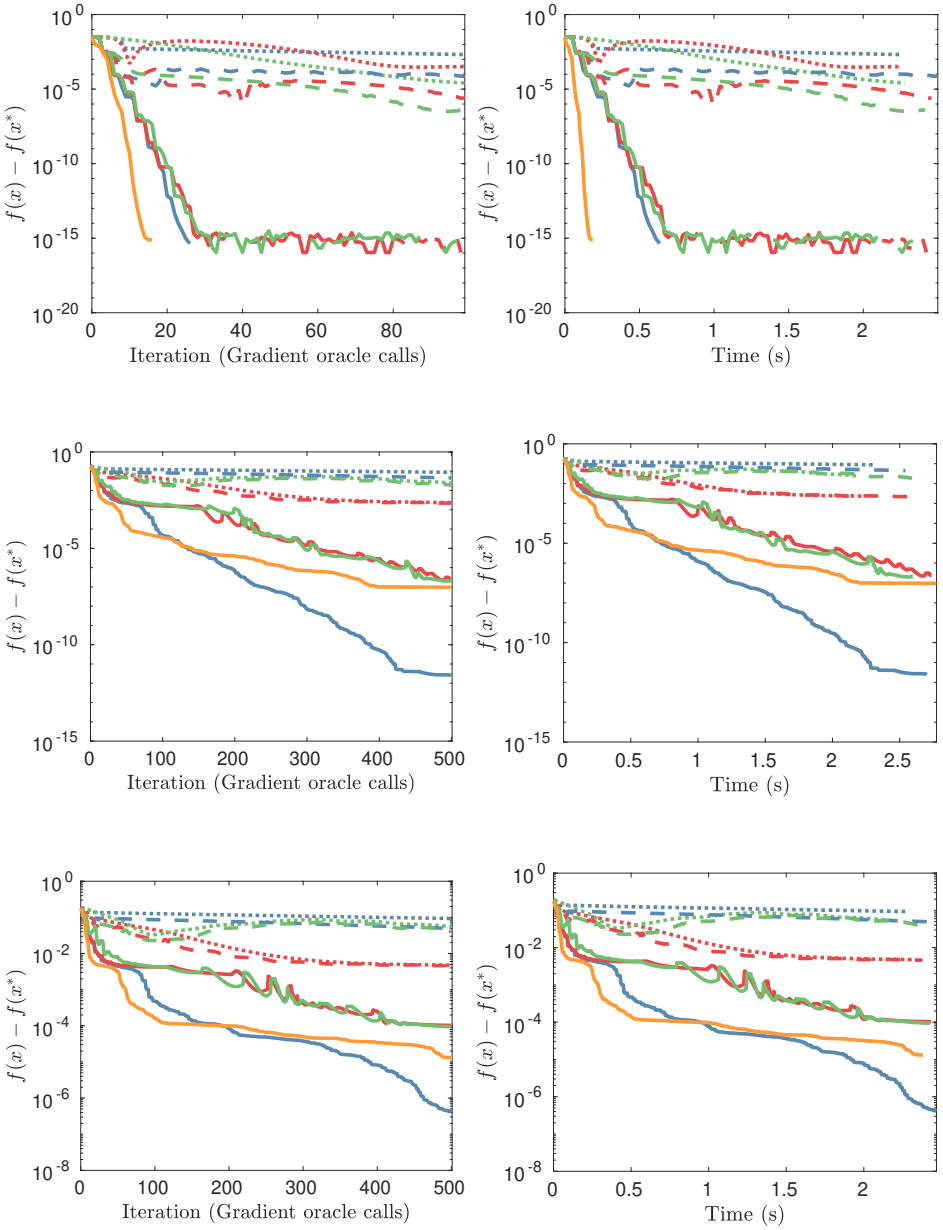
In addition, we run the ℓ-BFGS algorithm. We use three different regularization parameters to tune the condition number κ, which are *well conditioned* ($\kappa = 10^{-3}$), *normally conditioned* ($\kappa = 10^{-6}$) and *badly conditioned* ($\kappa = 10^{-9}$). In these experiments, for the RNA algorithm, we use as regularization parameter $\lambda = 10^{-8}\|\tilde{R}^T\tilde{R}\|$, and a window size of 10. Of course, these parameters can be fine-tuned to enjoy the best possible performance, but for simplicity we chose to keep these values fixed. For the ℓ-BFGS algorithm we also have a window size of 10 gradients, and a Wolfe-line-search.

We used an unified color and linestyle specification, summarized in Figure 4.3. Globally, Gradient algorithm is in blue, Nesterov algorithm in red (convex version) and green (strongly convex version), and ℓ-BFGS in orange. The original algorithm is in dotted line, while offline acceleration in dashed and online acceleration in plain line. Since ℓ-BFGS is sort of an acceleration of gradient descent, its representation is also in plain line.

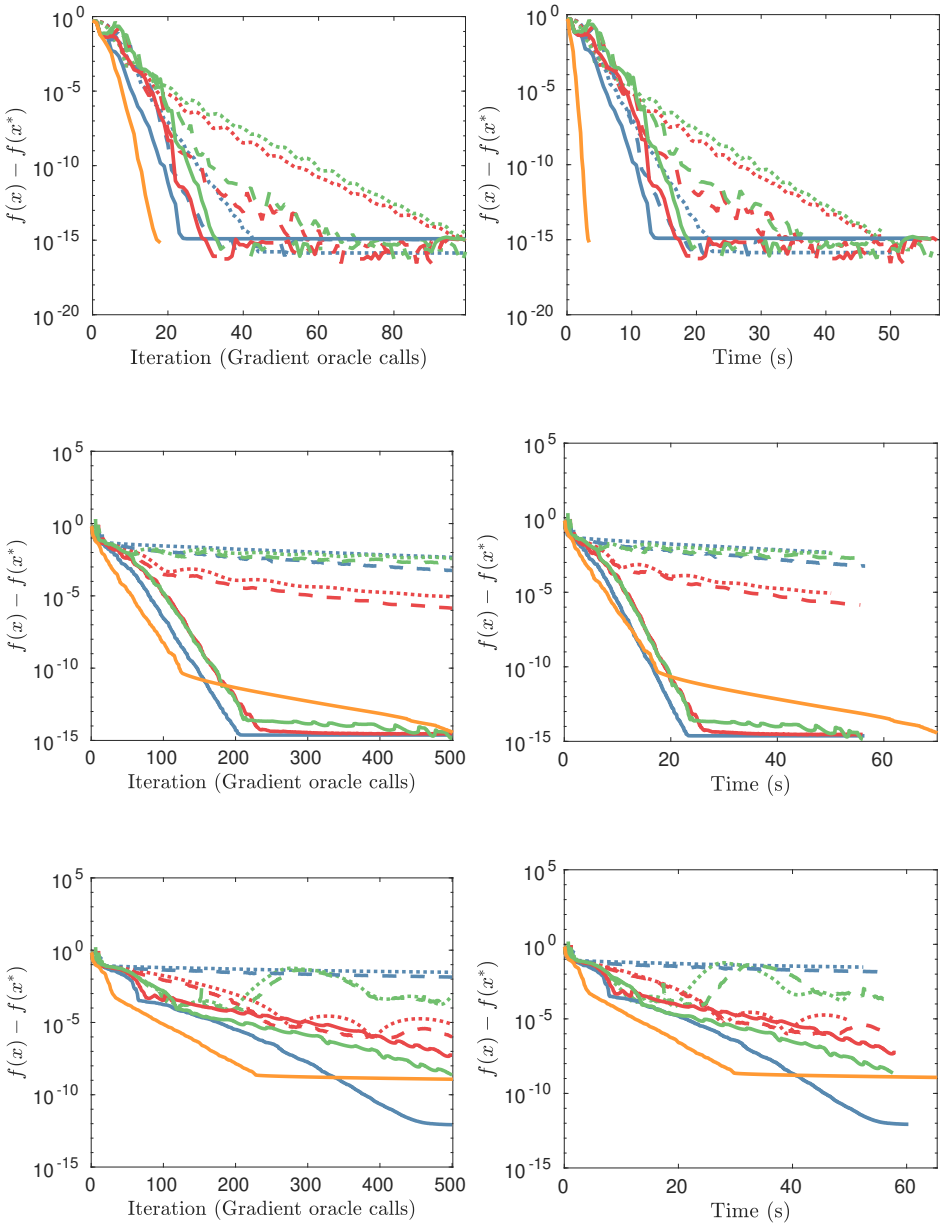
Dataset: sonar (Top to bottom: good, regular and bad condition number)



Dataset: madelon (Top to bottom: good, regular and bad condition number)



Dataset: sido0 (Top to bottom: good, regular and bad condition number)



4.3.3 Observations

In the previous experiments, we can observe that online acceleration (as well as BFGS) is always beneficial in comparison with the original optimization method, and globally, BFGS and RNA are competitive with each other, with a slight advantage to BFGS in the previous experiments. However, it is complicated to compare the two since there is a line-search subroutine in BFGS to ensure its convergence, while in RNA there is only one comparison between function values. In addition, there is no (good) global convergence rate for ℓ-BFGS, while Algorithm (5) preserves the rate.

Surprisingly, accelerating gradient descent with RNA looks to be a better strategy than accelerating Nesterov's method. In our experiments, in all cases, RNA combined with gradient descent always performs better than RNA with Nesterov, but the difference is quite small. On the other hand, accelerating an accelerated method guarantees an optimal rate of convergence. So acceleration with good global convergence rate comes at the cost of (sometimes) worse numerical convergence.

The claim that RNA makes momentum methods adaptive to the strong convexity parameter is verified in practice. Indeed, if we compare the two variants of Nesterov's method without acceleration, we have a big gap in the convergence rate (both numerically and theoretically). However, once they are accelerated with RNA, they present a similar rate of convergence.

Even if we have some extra computation effort, the difference in time between the original and accelerated version of an algorithm is negligible. It can be explained that we have a sum of function in our experiments, making the oracle call expensive. In comparison, RNA only needs to solve an $N \times N$ (where $N = 10$) system of linear equations. We can even speed up RNA using a recursive algorithm for solving this system, if needed.

In summary, online acceleration never hurts and often improves performance. Even from the theoretical point of view, in the worst case we multiply the number of call to $f(x)$ by two for getting the same result. It presents a similar numerical performance as BFGS despite the absence of line-search, but in addition a rate of convergence is guaranteed. Using online acceleration on accelerated methods ensures optimal rates, but come at the cost of slightly worse numerical performance.

4.4 Acceleration of Stochastic Algorithms

4.4.1 Introduction

We focus on the problem

$$\min_{x \in \mathbb{R}^d} f(x) \tag{4.6}$$

where $f(x)$ is a smooth and strongly convex function with respect to the Euclidean norm. We consider a stochastic first-order oracle, which gives a noisy estimate of the gradient of $f(x)$, with

$$\nabla_\varepsilon f(x) = \nabla f(x) + \varepsilon, \tag{4.7}$$

where ε is a noise term with bounded variance. This is the case for example when f is a sum of strongly convex functions, and we only have access to the gradient of one randomly selected function. Stochastic optimization (4.7) is typically challenging as classical algorithms are not convergent for a constant step size (for example, gradient descent or Nesterov's accelerated gradient). Even the averaged version of stochastic gradient descent with constant step size does not converge to the solution of (4.6), but to another point whose proximity to the real minimizer depends of the step size [49, 48].

When f is a finite sum of N functions, then algorithms such as SAG [63], SAGA [18], SDCA [69] and SVRG [38] accelerate convergence using a variance reduction technique akin to control variate in Monte-Carlo methods. Their rate of convergence depends on $1 - \mu/L$ and thus does exhibit an accelerated rate on par with the deterministic setting (in $1 - \sqrt{\mu/L}$), where L is the smoothness constant and μ the strong convexity constant of f. Recently a generic acceleration algorithm called Catalyst [45], based on the proximal point methods improved this rate of convergence, at least in theory. Unfortunately, numerical experiments show this algorithm to be conservative, thus limiting practical performances. On the other hand, recent papers, for example [70] (Accelerated SDCA) and [2] (Katyusha), propose algorithms with accelerated convergence rates, if the strong convexity parameter is given.

When f is a quadratic function then averaged SGD converges, but the rate of decay of initial conditions is very slow. Recently, some results have focused on accelerated versions of SGD for quadratic optimization, showing that with a two step recursion it is possible to enjoy both the optimal rate for the bias term and the variance [26], given an estimate of the ratio between the distance to the solution and the variance of ε.

In this section, we will analyse the impact of nonlinear acceleration to stochastic algorithms. We hope this strategy will still be adaptive to the strong convexity constant, whose online estimation is still a challenge, even in the deterministic case [25].

Before going in the numerical part, we analyse quickly the impact of stochastic noise on the extrapolation.

4.4.2 Acceleration with Noisy Iterates

We have seen here that in addition with non-linearities, a stochastic noise is present in (Comb. Pert. LFPI). We already analyzed the impact of non-linearities in the previous section, so here we focus only on stochastic noises with zero-mean and variance σ^2. In this case,

Proposition 3.3.11 becomes

$$\mathbb{E}\left[\|x_{\text{extr}} - x^*\|\right] \leq \|x_0 - x^*\| \frac{1}{\kappa}\left(S_\kappa\left(N, \gamma\right) + \frac{\gamma\sigma}{\|x_0 - x^*\|}\left(1 + \frac{1}{\kappa}\right)\right).$$

The rate of convergence of acceleration thus depends on the ratio $\frac{\sigma}{\|x_0 - x^*\|}$, close to the concept of *signal to noise* ratio. In particular, if σ is *dependent* of $\|x_0 - x^*\|$, a typical behavior when using SGD on quadratics, then acceleration *does no converge* when this ratio becomes too large. However, we can expect a better convergence for early iterations (which is the part where SGD generalizes the best [48]) because $\|x_0 - x^*\|$ will be large in comparison with σ.

On the other hand with variance-reduction techniques, such as SAGA and SVRG, σ asymptotically goes to zero. To reduce the ratio $\epsilon/\|x_0 - x^*\|$, it suffices to sufficiently wait between two "iterations" to reduce this ratio, i.e., considering x_{i+1} to be

$$x_{i+1} = g(g(\dots g(x_i)\dots)),$$

for a large number of calls of g.

We will make a coarse, non totally rigorous, analysis to give an intuition why waiting several iteration leads to smaller variance. For example, consider the case where the standard deviation of x_i is proportional to the distance to the optimum, i.e.,

$$\mathbb{E}[\|x_i - \mathbb{E}[x_i]\|] = \sigma\|\mathbb{E}[x_i] - x^*\|.$$

In addition, assume the method has a certain rate of convergence $1 - r$,

$$\mathbb{E}[\|x_{i+1} - x^*\|] \leq (1 - r)\mathbb{E}[\|x_i - x^*\|]$$

Then, if we wait s iterations,

$$\mathbb{E}[\|x_s - x^*\|] \leq (1 - r)^s\mathbb{E}[\|x_0 - x^*\|]$$

which means that the variance of x_s is bounded by

$$\mathbb{E}[\|x_s - \mathbb{E}[x_s]\|] \leq \sigma\mathbb{E}[\|x_i - x^*\|] \leq (1 - r)^s\sigma\|x_0 - x^*\|,$$

which decreases over time. The upper bound for the convergence rate becomes, in that case,

$$\mathbb{E}\left[\|x_{\text{extr}} - x^*\|\right] \leq \|x_0 - x^*\| \frac{1}{\kappa}\left(S_\kappa\left(N, \gamma\right) + \gamma(1 - r)^s\sigma\left(1 + \frac{1}{\kappa}\right)\right).$$

With s sufficiently large, we can make the right hand side arbitrary small. Of course, we have

to find the best trade-off between accelerating as much as we can, and waiting s non-accelerated iterations.

For simplicity, we consider in the rest of this section $s = m$ (i.e., we wait one epoch), the number of data points in the data set, because it seems to be sufficiently large in our numerical experiments. Again, for simplicity, we consider x_{i+1} to be actually $x_{i+1} = g^m(x_i)$, which means that "one iteration" for RNA corresponds to m stochastic iterations.

4.5 Numerical Experiments (Stochastic)

We focus on the composite problem

$$\min_{x \in R^d} f(x) = \sum_{i=1}^{m} \frac{1}{m} f_i(x) + \frac{\tau}{2}\|x\|^2,$$

where f_i are convex and L-smooth functions and τ is the regularization parameter. We will use classical methods for minimizing $f(x)$ such as SGD (with fixed step size), SAGA [18], SVRG [38], and also the accelerated algorithm Katyusha [2]. We will compare their performances with and without the (potential) acceleration provided by Algorithm 3. We set $\lambda = 10^{-8}\|\tilde{R}^T\tilde{R}\|$ and $N = 10$ for all the experiments.

4.5.1 Comparison Between Acceleration Strategies

In order to balance the complexity of the extrapolation algorithm and the optimization method we wait several data queries before adding the current point (the "snapshot") of the method to the sequence. Indeed, the extrapolation algorithm has a complexity of $O(N^2 d)$, where N is small (10 in our experiments). If we wait at least $O(d)$ updates, then the extrapolation method is of the same order of complexity as the optimization algorithm. A similar technique is used in [46].

- **SGD.** We add the current point after m data queries (i.e. one epoch). The residual is considered to be $x_{i+1} - x_i$ where x_i is the variable before the epoch and x_{i+1} the variable after the epoch.
- **SVRG.** We compute the gradient exactly, then perform m queries (the inner-loop of SVRG). The iterate x_i corresponds to the point before computing the full gradient, and x_{i+1} is the point computed after one iteration of the outer-loop. This means one iteration of SVRG is twice the cost of one iteration of SGD.
- **SAGA.** The main difference between SAGA and SGD is the presence of a table of computed gradients, but this does not matter when using RNA algorithm. We accelerate it exactly as SGD, and we do not touch the table of gradient between two accelerations.
- **Katyusha.** We compute the gradient exactly, then perform $4m$ gradient calls (the inner-loop of Katyusha). We will not accelerate Katyusha with RNA, as its expression does not

match perfectly (Comb. Pert. LFPI).

We compare these various methods for minimizing a logistic loss on Sonar [31], Madelon [34] and sido0 [33], with several condition numbers κ: well ($\kappa = 100/m$), moderately ($\kappa = 1/m$) and badly ($\kappa = 1/100m$) conditioned.

Like before, we compare the original optimization algorithm with its offline and online acceleration using RNA:

- **(V)** Vanilla: we run the original algorithm with no acceleration.
- **(P)** Offline version, in parallel: we run the RNA algorithm on the side (Algorithm 3), with no interaction on the original optimization method.
- **(O)** Online: we use the online version of RNA (Algorithm 4). We do not use Algorithm 5, because a comparison between function values (i.e., computing $h(x)$) is costly in stochastic optimization as it requires one data pass.

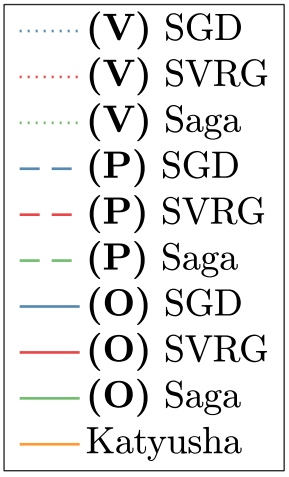
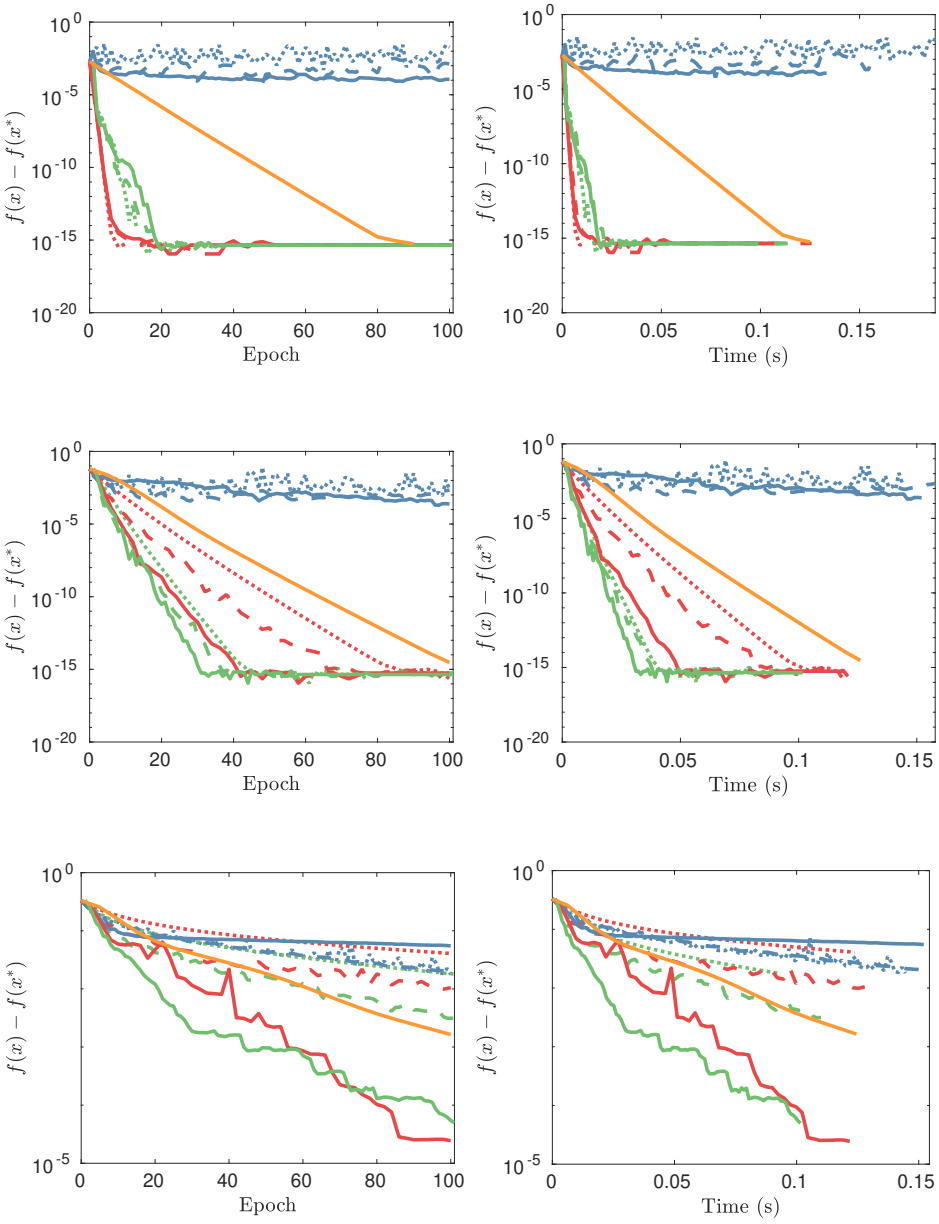
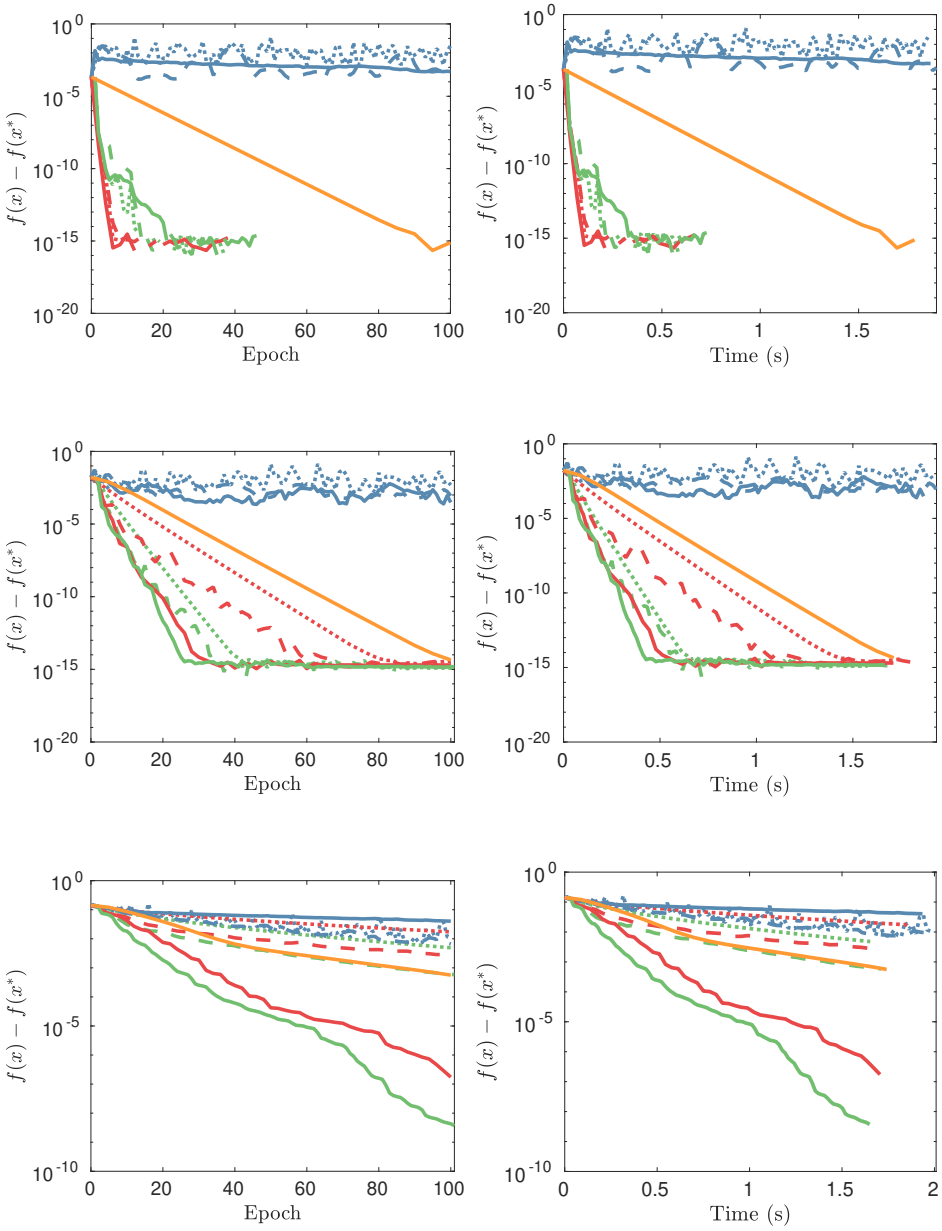


Figure 4.3: Legend for numerical experiments on stochastic algorithms

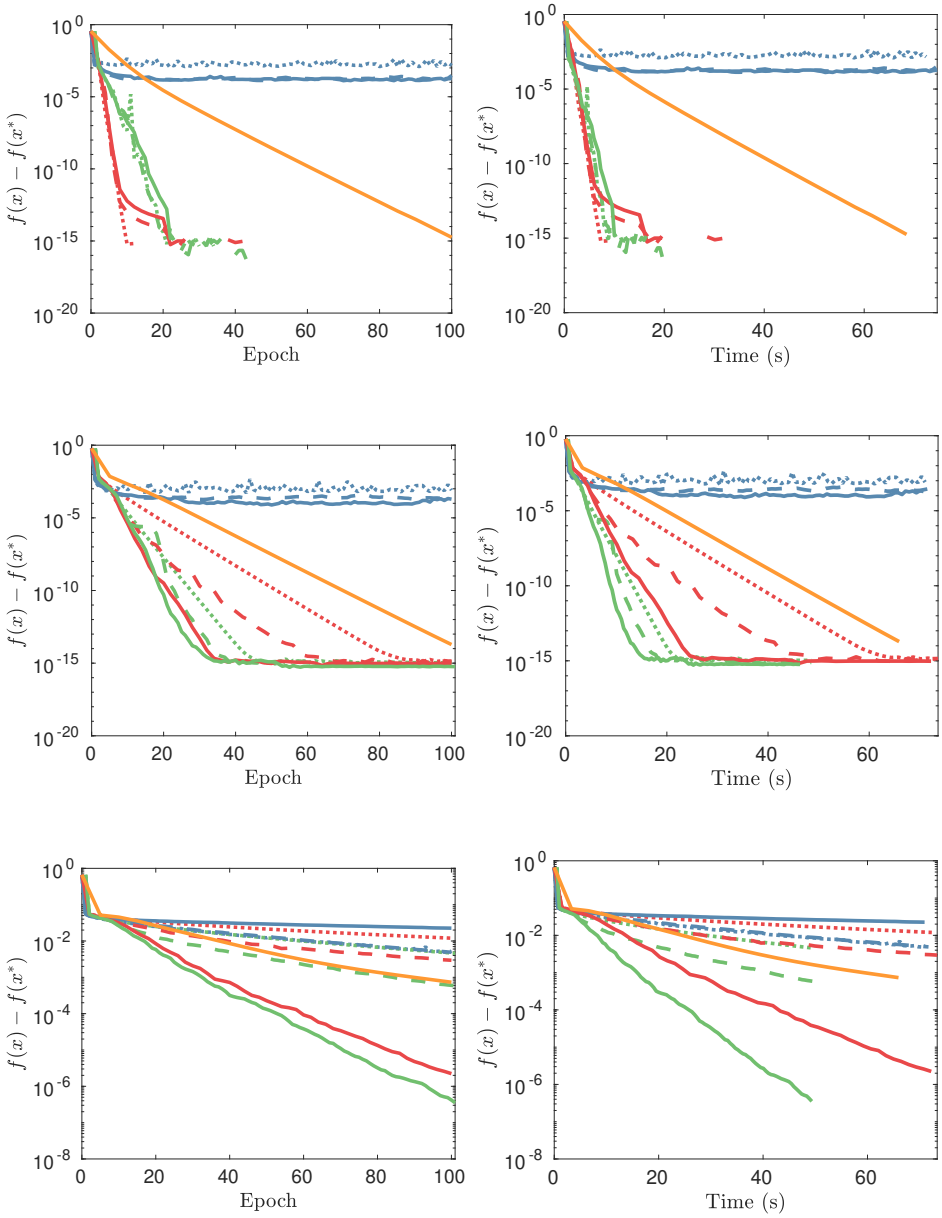
Dataset: sonar (Top to bottom: good, regular and bad condition number)



Dataset: madelon (Top to bottom: good, regular and bad condition number)



Dataset: sido0 (Top to bottom: good, regular and bad condition number)



4.5.2 Observations

In the previous figures, we clearly see that both SGD and its accelerated versions do not converge. This is mainly due to the fact that the variance is contant over iterations as we do not perform averaging. In addition, excepted for quadratic problems, the averaged version of SGD with constant step size does not converge with arbitrary precision.

Because the iterates of SAGA and SVRG have low variance, their accelerated version converges faster to the optimum. Even if we can see that, in our experiments, SAGA performs usually better than SVRG, their online accelerated version is comparable.

When we accelerate SAGA and SVRG using the offline version of RNA, we obtain a method whose performance is better than Katyusha, but without the specification of the strong convexity parameter.

Like in the deterministic case, the online acceleration works better than its offline counterpart, itself improving the original optimization method, when we have a variance reduction technique.

4.6 Optimizing Convolutional Neural Networks

4.6.1 Introduction

Stochastic gradient descent is a popular and effective method to train neural networks [48, 19]. A lot of efforts have been invested in accelerating stochastic algorithms, in particular by deriving methods that adapt to the structure of the problem. Algorithms such as RMSProp [76] or Adam [40] are examples of direct modifications of gradient descent, and estimate some statistical momentum during optimization to speed up convergence. Unfortunately, these methods can fail to converge on some simple problems [61], and may fail to achieve state-of-the-art test accuracy on image classification problems. Other techniques have been developed to improve convergence speed or accuracy, such as adaptive batch-size for distributed SGD [32], or quasi-second-order methods which improve the rate of convergence of stochastic algorithms [9]. However, only a limited number of settings are covered by such techniques, which are not compatible with state-of-the-art architectures.

The approach we propose here is completely different as RNA algorithm is built on top of existing optimization algorithms. Classical algorithms typically retain only the last iterate or the average [60] of iterates as their best estimate of the optimum, throwing away all the information contained in the converging sequence of iterates. As it is highly wasteful from a statistical perspective, extrapolation schemes estimate the optimum of an optimization problem using a weighted average of the last iterates produced by an algorithm, where the weights depend on the iterates.

Network ensembling [82] can also be seen as combining several neural networks to improve convergence. However, contrary to our strategy, ensembling consists in training different networks *from different starting points* and then averaging the predictions or the parameters. Averaging the weights of successive different neural networks from SGD iterations has been studied in [37], and RNA can also be seen as an extension of this averaging idea for SGD, but with non-uniform adaptive weights.

Overall, nonlinear acceleration has marginal computational complexity. On neural network training problems, we will see the offline version improves both the test accuracy of early iterations, as well as the final accuracy with only minor modifications of existing learning pipelines. It

never hurts performance as it runs on top of the algorithm and does not affect iterations. Finally, the scheme produces smoother learning curves. When training neural networks, we observe in the experiments that the convergence speedup produced by acceleration is much more significant in early iterations, which means it could serve for "rapid prototyping" of network architectures, with significant computational savings.

4.6.2 Numerical Experiments

The following numerical experiments seek to highlight the potential benefits of RNA in its offline and online versions when applied to the gradient method (with or without momentum term). Since the complexity grows quadratically with the number N of points in the sequences $\{x_i\}$ and $\{y_i\}$, we will use RNA with a fixed window size ($N = 5$ instead of $N = 10$ used previously, because neural networks has millions of parameters) in all these experiments. These values are sufficiently large to show a significant improvement in the rate of convergence, but can of course be fine-tuned. For simplicity, like in the previous experiments we fix $\lambda = 10^{-8}\|\tilde{R}^T\tilde{R}\|_2$.

We now describe experiments with CNNs (Convolutional Neural Networks) for image classification. Because one stochastic iteration is not informative due to the noise, we refer to x_i as the model parameters (including batch normalization statistics) corresponding to the final iteration of the epoch i, and y_i the model parameters (including batch normalization statistics) corresponding to the model right before the first stochastic iteration of the epoch i (see Section 4.4.2 for more details). In this case, we do not have an explicit access to "$(x_i - y_{i-1})$", so we will estimate it during the stochastic steps. Let $y_i^{(t)}$ be the parameters of the network at epoch i after t stochastic iterations, and $x_i^{(t+1)}$ be the parameters after one stochastic gradient step. Then, for a data set of size m,

$$x_i - y_{i-1} \approx \frac{1}{m}\sum_{t=1}^{m}(x_i^{(t+1)} - y_i^{(t)}) = -h\frac{1}{m}\sum_{t=1}^{m}\nabla f(y_i^{(t)}),$$

where h is the stepsize, also called learning rate in the community of deep neural networks. This means the matrix $\tilde{R}$ in Algorithm 3 will be the matrix of (estimated) gradients.

Because the learning curve is highly dependent on the learning rate schedule, we decided to use a linearly decaying learning rate to better illustrate the benefits of acceleration, even if acceleration also works with a constant learning rate schedule (see [67] and Figure 4.6). In all our experiments, until epoch T, the learning rate decreases linearly from an initial value h_0 to a final value h_T, with

$$h_k = h_0 + (k/T)(h_T - h_0). \tag{4.8}$$

We then continue the optimization during 10 additional epochs using h_T to stabilize the curve. We summarize the parameters used for the optimization in Table 4.1.

	h_0	h_T	momentum
SGD and Online RNA	1.0	0.01	0
SGD + momentum	0.1	0.001	0.9

Table 4.1: Parameters used in (4.8) to generate the learning rate for optimizers. We used the same setting for their accelerated version with RNA.

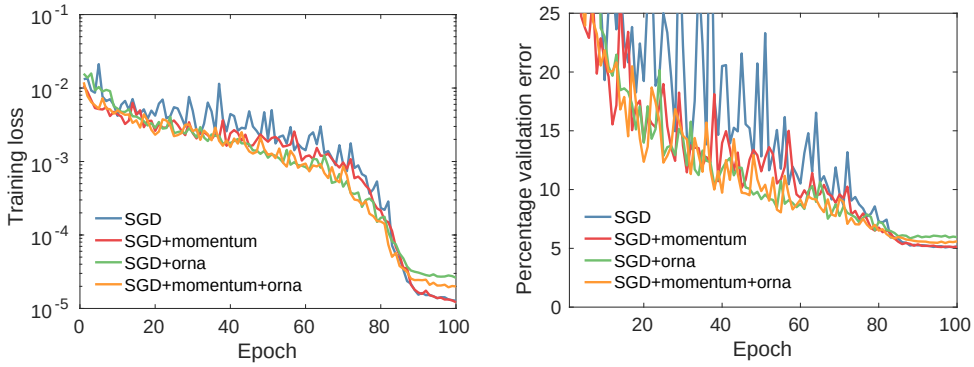


Figure 4.4: Training a Resnet-18 on Cifar10 for 100 epochs with SGD and the online RNA (Algorithm 4). On the left, the evolution of the loss value. On the right, the error percentage on the testing set. We see the online version does not improve the performance of SGD. This may be explained by the presence of the batchnorm module. The presence of a statistical estimation *while* the optimization is running is not taken into account in the iteration model (Comb. Pert. LFPI), and thus makes theoretical analysis more complex. In addition, because we are optimizing a nonconvex function, and the online version interferes with the optimization scheme, the final performance is not guaranteed to be as good as the non-accelerated version of the optimizer.

CIFAR10: Online Acceleration

CIFAR-10 is a standard 10-class image dataset comprising $5{\cdot}10^4$ training samples and 10^4 samples for testing. Except for the linear learning rate schedule above, we follow the standard practice for CIFAR-10. We applied the standard augmentation via padding of 4 pixels. We trained the networks VGG19, ResNet-18 and DenseNet121 during 100 epochs ($T = 90$) with a weight decay of $5 \cdot 10^{-4}$. The loss function is the cross-entropy.

We observe in Figure 4.4 that the online version does not perform as well as in the convex case. More surprisingly, it is outperformed by its offline version (Figure 4.5) which computes the iterates on the side.

CIFAR10: Offline Acceleration

In fact, the offline experiments detailed in Figure 4.5 exhibit much more significant gains. It produces a similar test accuracy, and the offline version converges faster than SGD, especially for early iterations. We reported speedup factors to reach a certain tolerance in Tables 4.2, 4.3

and 4.4. This suggests that the offline version of RNA is a good candidate for training neural networks, as it converges faster while guaranteeing performance *at least* as good as the reference algorithm.

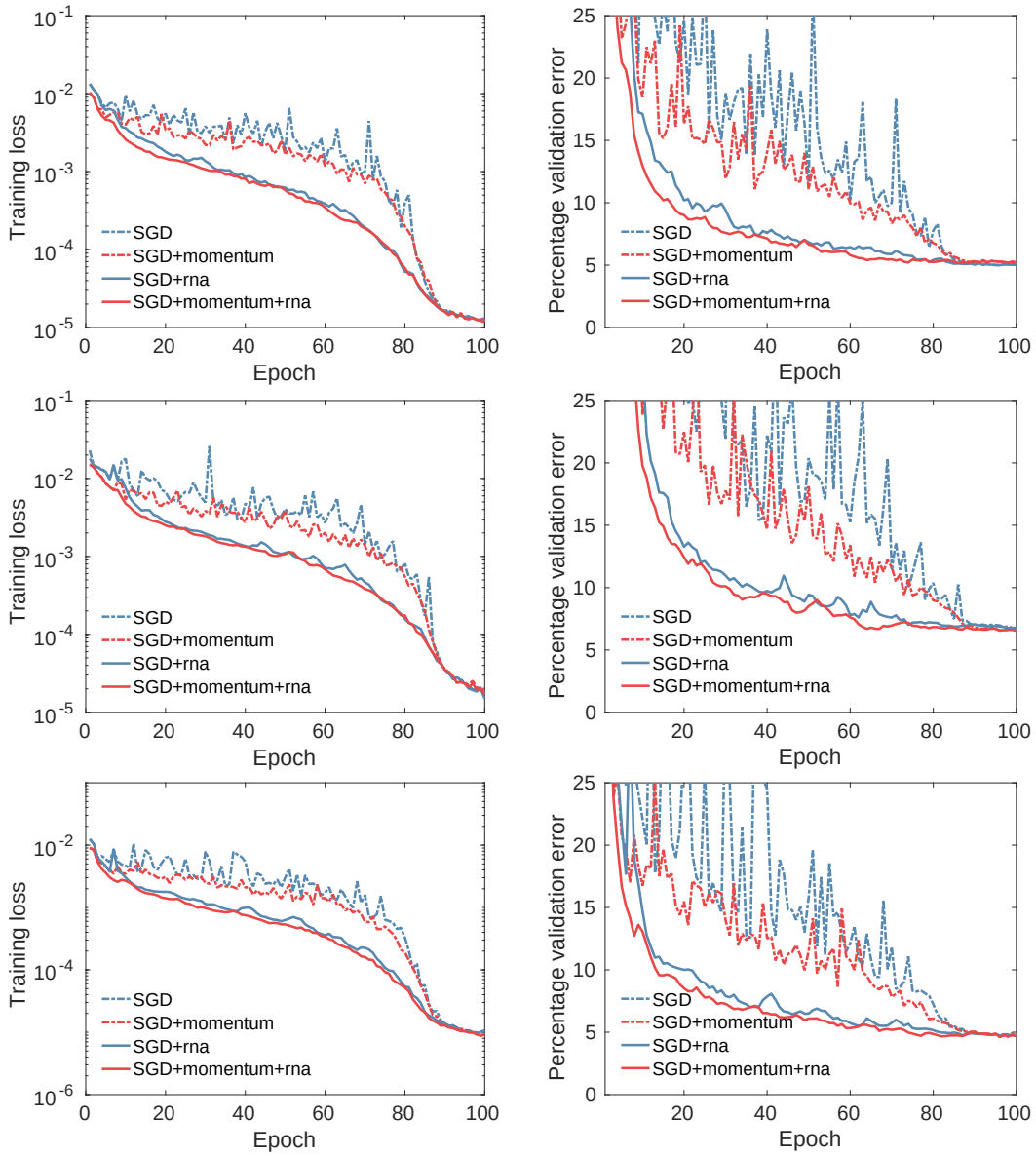


Figure 4.5: Training a Resnet-18 (*top*), VGG19 (*middle*) and a DenseNet-121 (*bottom*) on Cifar10 for 100 epochs. SGD with and without momentum, and their off-line accelerated versions with a window size of 5. On the left, the evolution of the loss value. On the right, the error percentage on the testing set.

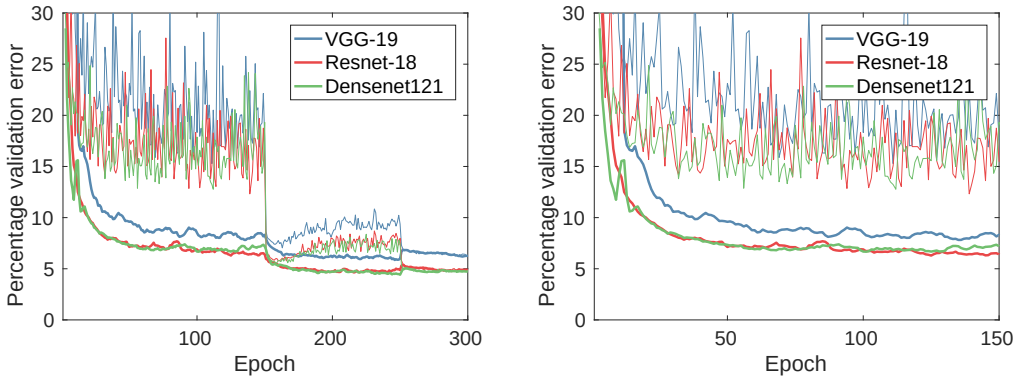


Figure 4.6: Prototyping networks: acceleration of SGD (bottom) gives a smoother convergence, producing a clearer ranking of architectures, earlier (flat learning rate). Right plot zooms on left one. The thin line corresponds to SGD, the bold to RNA+SGD.

Tolerance	SGD	SGD+momentum	SGD+RNA	SGD+momentum+RNA
5.0%	68 (0.87×)	59	21 (2.81×)	**16** (3.69×)
2.0%	78 (0.99×)	77	47 (1.64×)	**40** (1.93×)
1.0%	82 (1.00×)	82	67 (1.22×)	**59** (1.39×)
0.5%	84 (1.02×)	86	75 (1.15×)	**63** (1.37×)
0.2%	86 (1.13×)	97	**84** (1.15×)	85 (1.14×)

Table 4.2: Number of epochs required to reach the best test accuracy + *Tolerance*% on CIFAR10 with a Resnet18, using several algorithms (best accuracy is 5% here). The speed-up compared to the SGD+momentum baseline is in parenthesis.

Tolerance	SGD	SGD+momentum	SGD+RNA	SGD+momentum+RNA
5.0%	69 (0.87×)	60	26 (2.31×)	**24** (2.50×)
2.0%	83 (0.99×)	82	52 (1.58×)	**45** (1.82×)
1.0%	84 (1.02×)	86	71 (1.21×)	**60** (1.43×)
0.5%	89 (0.98×)	87	73 (1.19×)	**62** (1.40×)
0.2%	**N/A**	90	99 (0.90×)	**63** (1.43×)

Table 4.3: Number of epochs required to reach the best test accuracy + *Tolerance*% on CIFAR10 with VGG19. Here, the best accuracy is 6.54%

Imagenet

Here, we apply the RNA algorithm to the standard ImageNet dataset. We trained the networks during 90 epochs ($T = 80$) with a weight decay of 10^{-4} (it corresponds to add a ℓ_2 regularization in the objective). We reported the test accuracy on Figure 4.7 for the networks ResNet-50 and ResNet-152. We only tested the offline version of RNA here, because in previous experiments it gives better result than its online counterpart.

We again observe that the offline version of Algorithm 3 improves the convergence speed

Tolerance	SGD	SGD+momentum	SGD+RNA	SGD+momentum+RNA
5.0%	65 (0.86×)	56	22 (2.55×)	**13** (4.31×)
2.0%	80 (0.98×)	78	45 (1.73×)	**38** (2.05×)
1.0%	83 (1.00×)	83	60 (1.38×)	**56** (1.48×)
0.5%	87 (0.99×)	86	80 (1.08×)	**66** (1.30×)
0.2%	92 (1.01×)	93	86 (1.08×)	**75** (1.24×)

Table 4.4: Number of epochs required to reach the best test accuracy + *Tolerance*% on CIFAR10 with DenseNet121. Here, the best accuracy is 4.62%.

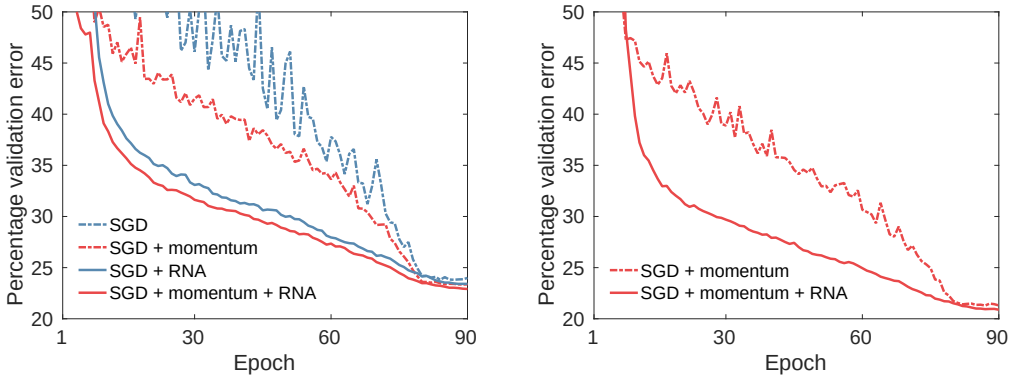


Figure 4.7: Training a Resnet-52 (*left*) and ResNet-152 (*right*) on validation ImageNet for 90 epochs using SGD with and without momentum, and their off-line accelerated versions.

	Pytorch	SGD	SGD+mom.	SGD+RNA	SGD+mom.+RNA
Resnet-50	23.85	23.808	23.346	23.412 (-0.396%)	**22.914** (-0.432%)
Resnet-152	21.69	N/A	21.294	N/A	**20.884** (-0.410%)

Table 4.5: Best validation top-1 error percentage on ImageNet. In parenthesis the improvement due to RNA. The first column corresponds to the performance of Pytorch pre-trained models.

of SGD with and without momentum. In addition, we show a substantial improvement of the accuracy over the non-accelerated baseline. The improvement in the accuracy is reported in Figure 4.5. Interestingly, the resulting training loss is smoother than its non accelerated counterpart, which indicates a noise reduction.

4.7 Conclusion and Perspectives

In this chapter, we analyzed the effect of acceleration on several optimization algorithms, in the deterministic and stochastic setting and even for non-convex neural networks. In the deterministic case, we studied the local rate of convergence, and we deduced that when $\lambda = O(D^{2(1+a)})$ with $a \in]0, 1[$ where D is an upper bound for $\|x_0 - x^*\|$, then we recover asymptotically an

optimal rate of convergence when $D \to 0$.

In the case of stochastic algorithm, we noticed that the rate mostly depends on the ratio between the noise variance and the distance to the optimum, which is sort of the inverse of the signal to noise ratio. This means acceleration cannot work on algorithm with large variance, such as SGD. However, if the distance between two stochastic iteration is big enough, then we can accelerate an optimization algorithm with variance-reduction technique such as SAGA or SVRG. Waiting several stochastic iteration is also beneficial since it allows to balance the time taken by the algorithm's iterations and RNA computation.

In the two previous cases, offline and online acceleration improve the numerical performance of optimization algorithms when minimizing a logistic loss, whatever the condition number of the problem is. Even compared with optimal methods, such as Nesterov's accelerated gradient (for deterministic optimization) and Katyusha (Stochastic optimization), the online RNA algorithm is much faster, due to its adaptivity to the problem's structure.

In the case of neural network, the conclusion is more mitigated, as the online version does not works as well as expected. In particular due to non-convexity, high dimension and statistical estimations during the optimization process (batchnorm modules), convonlutional neural networks are extremely hard to optimize. This may explain why online acceleration does not perform as well as in the convex case.

On the contrary to its online counterpart, offline acceleration preserves the performance as is does not interact with the optimization scheme. Surprisingly, this gives good acceleration results on neural networks. Even after a few number of iteration, the prediction accuracy of the extrapolated model is close to be optimal.

Here, we did not explore the limits of extrapolation. For example, is it possible to design an accelerated method for non-smooth problem? Even if some numerical experiments have been conducted in [64] on Lasso or dual-SVM, also in [46] on dual-Lasso, the theoretical background is still incomplete, since non-smooth problems are inherently non-differentiable.

Also, the similarity between BFGS and RNA performance may mean there is a link between the two methods. In fact, in [24] is studied several classes of non-linear acceleration methods, such as Anderson Acceleration (highly related to RNA) and Broyden's family (linked with BFGS). A deeper analysis may lead to new convergence results.

Chapter 5

Conclusion and Perspectives

In this thesis, we studied acceleration under different forms. First, we analyzed existing accelerated algorithms under another point of view, linking them with integration methods for the gradient flow. In particular, we showed that many existing optimization methods are special instances of standard integration schemes. If we assume classical conditions, such as consistency or stability, we showed that accelerated algorithms present larger step size, thus integrating (and converging) faster. Second, we modified and studied an extrapolation algorithm for vector sequences (Anderson Acceleration and Minimal Polynomial Method) and extended the analysis to optimization algorithms. We showed that this scheme, when regularized, improves significantly optimization methods performance, even when they are already accelerated. This is explained by its adaptivity to the local structure of the problem.

In the future, we hope that our links with integration methods can be extended. In particular, we are able to design methods with optimal rates, but only for quadratics. The extension to nonlinear function can be eventually done by studying G-stability [16, 14], which generalizes the study of integration schemes to smooth and strongly convex functions. On the other hand, strong convexity is a key component in our approach, and moving to convex function can be an interesting extension.

The analysis of the regularized nonlinear acceleration algorithm, even generic, does not depend on the problem's structure and the convergence bound is very conservative. If we introduce strong convexity and smoothness sooner in the analysis, it may be possible to derive more accurate convergence bounds. In addition, we need to use a regularized version of Chebyshev polynomial, whose exact formula is unknown. We also assumed the operator which generates the sequences to be symmetric. Considering a non-symmetric operator (for example when using primal-dual algorithms) can be interesting. Finally, in other experiments, the efficiency of RNA on non-smooth problem looks promising. However our analysis does not hold because by essence non-smooth problems are not differentiable. Using other arguments, it might be possible to show that generic acceleration is also achievable for sharp problems.

Bibliography

[1] A. C. Aitken. On bernoulli's numerical solution of algebraic equations. *Proceedings of the Royal Society of Edinburgh*, 46:289–305, 1927.

[2] Z. Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *arXiv preprint arXiv:1603.05953*, 2016.

[3] Z. Allen Zhu and L. Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. In *Proceedings of the 8th Innovations in Theoretical Computer Science*, ITCS 17, 2017.

[4] D. G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

[5] U. M. Ascher, S. J. Ruuth, and B. T. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM Journal on Numerical Analysis*, 32(3):797–823, 1995.

[6] A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

[7] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[8] A. Ben-Tal and A. Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[9] R. Bollapragada, D. Mudigere, J. Nocedal, H.-J. M. Shi, and P. T. P. Tang. A progressive batching L-BFGS method for machine learning. *arXiv preprint arXiv:1802.05374*, 2018.

[10] J. Bolte, A. Daniilidis, and A. S. Lewis. The Łojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems. *SIAM Journal on Optimization*, 17(4):1205–1223, 2007.

[11] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[12] C. Brezinski. *Accélération de la convergence en analyse numérique*, volume 584. Springer, 2006.

[13] S. Bubeck, Y. Tat Lee, and M. Singh. A geometric alternative to Nesterov's accelerated gradient descent. *ArXiv e-prints*, jun 2015.

[14] J. Butcher. Thirty years of g-stability. *BIT Numerical Mathematics*, 46(3):479–489, 2006.

[15] S. Cabay and L. Jackson. A polynomial extrapolation method for finding limits and antilimits of vector sequences. *SIAM Journal on Numerical Analysis*, 13(5):734–752, 1976.

[16] G. Dahlquist. G-stability is equivalent toa-stability. *BIT Numerical Mathematics*, 18(4): 384–401, 1978.

[17] G. Dahlquist. On one-leg multistep methods. *SIAM journal on numerical analysis*, 20(6): 1130–1138, 1983.

[18] A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[19] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, et al. Recent advances in deep learning for speech research at Microsoft. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8604–8608. IEEE, 2013.

[20] Y. Drori and M. Teboulle. Performance of first-order methods for smooth convex minimization: a novel approach. *Mathematical Programming*, 145(1-2):451–482, 2014.

[21] J. C. Duchi, S. Shalev-Shwartz, Y. Singer, and A. Tewari. Composite objective mirror descent. In *COLT*, pages 14–26, 2010.

[22] J. Durbin. The fitting of time-series models. *Revue de l'Institut International de Statistique*, pages 233–244, 1960.

[23] R. Eddy. Extrapolating to the limit of a vector sequence. *Information linkage between applied mathematics and industry*, pages 387–396, 1979.

[24] H.-R. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3):197–221, 2009.

[25] O. Fercoq and Z. Qu. Restarting accelerated gradient methods with a rough strong convexity estimate. *arXiv preprint arXiv:1609.07358*, 2016.

[26] N. Flammarion and F. Bach. From averaging to acceleration, there is only a step-size. In *Conference on Learning Theory*, pages 658–695, 2015.

[27] J. Frank, W. Hundsdorfer, and J. Verwer. On the stability of implicit-explicit linear multistep methods. *Applied Numerical Mathematics*, 25(2-3):193–205, 1997.

[28] W. Gautschi. *Numerical analysis*. Springer Science & Business Media, 2011.

[29] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[30] G. H. Golub and R. S. Varga. Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods. *Numerische Mathematik*, 3(1):157–168, 1961.

[31] R. P. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural networks*, 1(1):75–89, 1988.

[32] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[33] I. Guyon. Sido: A phamacology dataset. *URL: http://www. causality. inf. ethz. ch/data/SIDO. html*, 2008.

[34] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

[35] E. Hazan. Personal communication. 2014.

[36] G. Heinig and K. Rost. Fast algorithms for Toeplitz and Hankel matrices. *Linear Algebra and its Applications*, 435(1):1–59, 2011.

[37] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[38] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

[39] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pages 462–466, 1952.

[40] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] W. Krichene, A. Bayen, and P. L. Bartlett. Accelerated mirror descent in continuous and discrete time. In *Advances in neural information processing systems*, pages 2845–2853, 2015.

[42] J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[43] L. Lessard, B. Recht, and A. Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016.

[44] N. Levinson. The wiener rms error criterion in filter design and prediction, appendix b of wiener, n.(1949). *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*, 1949.

[45] H. Lin, J. Mairal, and Z. Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems*, pages 3384–3392, 2015.

[46] M. Massias, J. Salmon, and A. Gramfort. Celer: a fast solver for the lasso with dual extrapolation. In *International Conference on Machine Learning*, pages 3321–3330, 2018.

[47] M. Mešina. Convergence acceleration for the iterative solution of the equations x= ax+ f. *Computer Methods in Applied Mechanics and Engineering*, 10(2):165–173, 1977.

[48] E. Moulines and F. Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, 2011.

[49] A. Nedić and D. Bertsekas. Convergence rate of incremental subgradient algorithms. In *Stochastic optimization: algorithms and applications*, pages 223–264. Springer, 2001.

[50] A. Nemirovskii and Y. E. Nesterov. Optimal methods of smooth convex minimization. *USSR Computational Mathematics and Mathematical Physics*, 25(2):21–30, 1985.

[51] A. S. Nemirovskii and B. T. Polyak. Iterative methods for solving linear ill-posed problems under precise information. *ENG. CYBER.*, (4):50–56, 1984.

[52] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[53] Y. Nesterov. Squared functional systems and optimization problems. In *High performance optimization*, pages 405–440. Springer, 2000.

[54] Y. Nesterov. Gradient methods for minimizing composite objective function, 2007.

[55] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[56] Y. Nesterov. Universal gradient methods for convex optimization problems. *Mathematical Programming*, 152(1-2):381–404, 2015.

[57] Y. Nesterov and B. T. Polyak. Cubic regularization of Newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[58] P. A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000.

[59] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[60] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[61] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. In *International Conference on Learning Representations*, 2018.

[62] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

[63] M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, pages 1–30, 2013.

[64] D. Scieur, A. d'Aspremont, and F. Bach. Regularized nonlinear acceleration. In *Advances In Neural Information Processing Systems*, pages 712–720, 2016.

[65] D. Scieur, F. Bach, and A. d'Aspremont. Nonlinear acceleration of stochastic algorithms. In *Advances in Neural Information Processing Systems*, pages 3985–3994, 2017.

[66] D. Scieur, V. Roulet, F. Bach, and A. d'Aspremont. Integration methods and optimization algorithms. In *Advances in Neural Information Processing Systems*, pages 1109–1118, 2017.

[67] D. Scieur, E. Oyallon, A. d'Aspremont, and F. Bach. Nonlinear acceleration of CNNs. In *Workshop track of International Conference on Learning Representations (ICLR)*, 2018.

[68] D. Scieur, E. Oyallon, A. d'Aspremont, and F. Bach. Nonlinear acceleration of deep neural networks. *arXiv preprint arXiv:1805.09639*, 2018.

[69] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

[70] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *ICML*, pages 64–72, 2014.

[71] D. Shanks. Non-linear transformations of divergent and slowly convergent sequences. *Studies in Applied Mathematics*, 34(1-4):1–42, 1955.

[72] A. Sidi, W. F. Ford, and D. A. Smith. Acceleration of convergence of vector sequences. *SIAM Journal on Numerical Analysis*, 23(1):178–196, 1986.

[73] D. A. Smith, W. F. Ford, and A. Sidi. Extrapolation methods for vector sequences. *SIAM review*, 29(2):199–233, 1987.

[74] W. Su, S. Boyd, and E. Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pages 2510–2518, 2014.

[75] E. Süli and D. F. Mayers. *An introduction to numerical analysis*. Cambridge university press, 2003.

[76] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[77] E. E. Tyrtyshnikov. How bad are Hankel matrices? *Numerische Mathematik*, 67(2):261–269, 1994.

[78] A. Wibisono, A. C. Wilson, and M. I. Jordan. A variational perspective on accelerated methods in optimization. *Proceedings of the National Academy of Sciences*, page 201614734, 2016.

[79] A. C. Wilson, B. Recht, and M. I. Jordan. A Lyapunov analysis of momentum methods in optimization. *arXiv preprint arXiv:1611.02635*, 2016.

[80] P. Wynn. On a device for computing the e m (s n) transformation. *Mathematical Tables and Other Aids to Computation*, pages 91–96, 1956.

[81] G. Zhang and A. Xiao. Stability and convergence analysis of implicit–explicit one-leg methods for stiff delay differential equations. *International Journal of Computer Mathematics*, 93(11):1964–1983, 2016.

[82] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263, 2002.

Résumé

Dans de nombreux domaines, comme par exemple l'optimisation, la performance d'une méthode est souvent caractérisée par son taux de convergence. Cependant, accélérer un algorithme requiert une certaine connaissance de la structure du problème et de telles améliorations sont le fruit d'une étude au cas-par-cas. De nombreuses techniques d'accélération ont été développées ces dernières décennies et sont maintenant massivement utilisées. En dépit de leur simplicité, ces méthodes sont souvent basées sur des arguments purement algébriques et n'ont généralement pas d'explications intuitives.

Récemment, de nombreux travaux ont été menés pour faire des liens entre les algorithmes accélérés et d'autres domaines scientifiques, comme par exemple avec la théorie du contrôle ou des équations différentielles. Cependant, ces explications reposent souvent sur des arguments complexes et la plupart utilise des outils non-conventionnels dans leur analyse.

Une des contributions de cette thèse est une tentative d'explication des algorithmes accélérés en utilisant la théorie des méthodes d'intégration, qui a été très étudiée et jouit d'une analyse théorique solide. En particulier, nous montrons que les méthodes d'optimisation sont en réalité des instances de méthode d'intégration, lorsqu'on intègre l'équation du flot de gradient. Avec des arguments standards, nous expliquons intuitivement l'origine de l'accélération.

De l'autre côté, les méthodes accélérées ont besoin de paramètres supplémentaires, en comparaison d'autres méthodes plus lentes, qui sont généralement difficiles à estimer. De plus, ces schémas sont construits pour une configuration particulière et ne peuvent pas être utilisés autre part.

Ici, nous explorons une autre approche pour accélérer les algorithmes d'optimisation, qui utilise des arguments d'accélération générique. En analyse numérique, ces outils ont été développés pour accélérer des séquences de scalaires ou de vecteurs, en construisant parallèlement une autre séquence avec un meilleur taux de convergence. Ces méthodes peuvent être combinées avec un algorithme itératif, l'accélérant dans la plupart des cas. En pratique, ces méthodes d'extrapolation ne sont pas tellement utilisées, notamment dû à leur manque de garanties de convergence et leur instabilité. Nous étendons ces méthodes en les régularisant, ce qui permettra une analyse théorique plus profonde et des résultats de convergence plus fort, en particulier lorsqu'elles sont appliquées à des méthodes d'optimisation.

Abstract

In many different fields such as optimization, the performance of a method is often characterized by its rate of convergence. However, accelerating an algorithm requires a lot of knowledge about the problem's structure, and such improvement is done on a case-by-case basis. Many accelerated schemes have been developed in the past few decades and are massively used in practice. Despite their simplicity, such methods are usually based on purely algebraic arguments and often do not have an intuitive explanation.

Recently, heavy work has been done to link accelerated algorithms with other fields of science, such as control theory or differential equations. However, these explanations often rely on complex arguments, usually using non-conventional tools in their analysis.

One of the contributions of this thesis is a tentative explanation of optimization algorithms using the theory of integration methods, which has been well studied and enjoys a solid theoretical analysis. In particular, we will show that optimization scheme are special instance of integration methods when integrating the classical gradient flow. With standard arguments, we intuitively explain the origin of acceleration.

On the other hand, accelerated methods usually need additional parameters in comparison with slower one, which are in most cases difficult to estimate. In addition, these schemes are designed for one particular setting and cannot be used elsewhere.

In this thesis, we explore a new approach for accelerating optimization algorithms, which uses generic acceleration arguments. In numerical analysis, these tools have been developed for accelerating sequences of scalars or vectors, by building on the side another sequence with a better convergence rate. These methods can be combined with an iterative algorithm, speeding it up in most cases. In practice, extrapolation schemes are not widely used due to their lack of theoretical guarantees and their instability. We will extend these methods by regularizing them, allowing a deeper theoretical analysis and stronger convergence results, especially when applied to optimization methods.