Geometric Numerical Integration and Optimization in Machine Learning

Ramiyou Karim Mache (karim.r.mache@aims-senegal.org)

African Institute for Mathematical Sciences (AIMS), Senegal

Supervised by: Pr. David Cohen (Umeå University, Sweden)

January 23, 2020

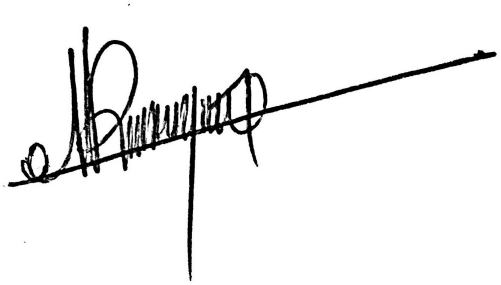
Submitted in partial fulfilment of the requirements for the award of Master of Science in Mathematical Sciences at AIMS Senegal

DECLARATION

This work was carried out at AIMS Senegal in partial fulfilment of the requirements for a Master of Science Degree.

I hereby declare that except where due acknowledgement is made, this work has never been presented wholly or in part for the award of a degree at AIMS Senegal or any other University.

Ramiyou Karim Mache



ACKNOWLEDGEMENTS

I am grateful to God for the good health, wellbeing and energy that were necessary to complete this work.

Foremost, I would like to express my deep and sincere gratitude to my supervisor Prof. David Cohen for the continuous support of my master thesis, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisor and mentor for my master thesis.

I would like to express my deep and sincere gratitude to the African Institute for mathematical Sciences (AIMS) for this opportunity. I thank the administrative staff of AIMS Senegal, Prof. Youssef Travely, Mr Amdou Cisse, Prof. Franck K. Mutombo, Mrs Layih Butake, Mr Diallo, Mrs Nathalie Texiera, Dr. Charle Kimpolo, for their effort during my study at AIMS.

I express my very profound gratitude to Dr.Takam Soh Patrice and Prof. Ayissi Raoul for recommending me to AIMS.

I express my very profound gratitude to all the tutors of AIMS Senegal for their assistance during my study and my research.I thank Dr. Ignace Aristid Minlend, Dr. Cheikh Birahim Ndao, Dr. Mohammed Akram, Mr. Michel Seck, for their particular assistance.

I am extremely grateful to my fellow students, for their team spirit and sharing.

I must express my very profound gratitude to my parents for their love, prayers, caring and sacrifices for educating and preparing me for my future.

I am very much thankful to my betrothed Makene L. Alma and the rest of my family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Last but not the least my thanks go to all my friends and all the people who have supported me to complete the research work directly or indirectly.

DEDICATION

Mother & Father,

whose affection, love, support, encouragement and prays of day and night make me able to get such success and honor.

Abstract

In the last years, machine learning has developed very quickly, and has become an indispensable part of our modern society. It has made many theoretical breakthroughs, and is widely applied in various fields. Optimization, as an important part of machine learning, has attracted much attention of researchers. With the exponential growth of data amount and the increase of model complexity, optimization methods in machine learning face more and more challenges. The most popular numerical algorithms for solving such optimization problems are gradient-based optimization algorithms, including gradient descent method, Nesterov's accelerated gradient descent method or stochastic gradient descent method. In this work, we give a brief overview of machine learning and describe how optimization problem appear in many field of machine learning and deep learning. We study some of the above gradient-based optimization algorithms in continuous-time perspective, establishing links between such methods and some special Ordinary Differential Equations (ODEs). We study the so-called Bregman Lagrangian, which generates a large class of accelerated methods in continuous time. The corresponding Hamiltonian dynamical systems allow us to apply a geometric numerical method, Leapfrog method, to solve convex optimization problems in machine learning. To explore the performance of such geometric numerical methods, we compare it to the three-step dynamical Nesterov discretization. From this comparison, we observed that the three-step dynamical Nesterov discretization converges quickly to the minimum for a good step size but diverges when this step size becomes large, whereas the geometric numerical scheme even converges slowly, and remain stable when the step size increases.

Keywords: Machine learning, Optimization, gradient-based optimization algorithms, Bregman Lagrangian, geometric numerical methods.

Contents

List of Figures

Notations

Here We provide some notations used in this document.

$\mathbb{N}$ is the set of natural numbers.

$\mathbb{R}$ is the set of real numbers.

The letter n stands for a natural number.

For a given function y, y' and y'' denote the first and second derivative of y respectively, and for $k \in \{3, \dots, n\}$, $y^{(k)}$ designs the kth derivative of the function y.

$\mathcal{C}^n(I)$ is the set of functions continuously differentiable up to order n on an interval I.

Id stands for the identity function from $\mathbb{R}^n$ to $\mathbb{R}^n$.

By a domain we understand any connected open subset of a finite-dimensional vector space.

We denote by $\mathcal{F}_L^1(\mathbb{R}^n)$ the class of functions that are convex with L-Lipschitz gradient. It means that,

$f \in \mathcal{F}_L^1(\mathbb{R}^n)$ if $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle$ and $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$, for all $x, y \in \mathbb{R}^n$. $\mathcal{F}_L^2(\mathbb{R}^n)$ denote the subclass of $\mathcal{F}_L^1(\mathbb{R}^n)$ consisting of the functions that have a L-Lipschitz continuous Hessian.

For $p = 1, 2$, $\mathcal{S}_{\mu,L}^p(\mathbb{R}^n)$ design the subclass of $\mathcal{F}_L^p(\mathbb{R}^n)$ consisting of the functions that are μ-strongly convex for some $0 < \mu \leq L$. It means that $f \in \mathcal{S}_{\mu,L}^p(\mathbb{R}^n)$ if $f \in \mathcal{F}_L^p(\mathbb{R}^n)$ and $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|$, for all $x, y \in \mathbb{R}^n$.

Introduction

Nowadays, machine learning has grown at a remarkable rate and has become an indispensable part of our modern society (LeCun et al. (2015)). Deep learning is a subfield of machine learning that allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. Machine learning and deep learning play a significant role in many fields such as machine translation, decision making, speech and image recognition, fraud detection, recommendation systems, self-driving car, etc. Most machine learning and deep leaning algorithms can be formulated as an optimization problem to find the extremum of an objective function or to find the minimum of a cost function, that can be convex or non-convex (Bottou (2012), Sun et al. (2019)).

The most useful numerical algorithms to solve such minimization problems are gradient-based optimization algorithms including gradient descent method, Nesterov's accelerated gradient descent method or stochastic gradient descent method. Some of these gradient-based methods can be viewed as a discretization of certain class of Ordinary Differential Equations (ODEs), this motivated the study of such methods in continuous-time perspective and then obtaining the algorithms via discretization (Su et al., Wibisono et al. (2016), Shi et al. (2019)). For instance, introduction of the so-called Bregman Lagrangian allows to generates a large class of accelerated methods in continuous time. This then allows to develop a whole family of numerical schemes with the same rate of convergence as Nesterov's method (Wibisono et al. (2016), Wilson et al. (2016)). The class of differential equations arising from the Bregman Lagrangian are special in the sense that the underlying Lagrangian and Hamiltonian pocess certain properties that need to be preserved by the numerical discretization. The field of geometric numerical integration shows that it is possible to preserve such properties when discretizing these differential equations (Hairer et al. (2006)).

In this work, we explain how optimization problems arise in machine learning and deep learning, establish links between gradient-based methods and special ODEs, and apply symplectic integration (a geometric numerical integrator) to gradient-based optimization.

The remainder of this thesis is organized as follow. Chapter 1 is devoted to basic concepts of ODEs, Hamiltonian systems and geometric numerical integrators. Chapter 2 explains how optimization problems arise in the fields of machine learning and deep learning. Chapter 3 establishes formally links between some gradient-based algorithms such as gradient descent, Polyak heavy ball, Nesterov's accelerated gradient descent, and some special ODEs. We study the accelerated methods from a continuous-time, variational point of view in which we explore the Bregman Lagrangian introduced by (Wibisono et al. (2016)), and find the corresponding Hamiltonian ODEs. To explore the performance and the stability of symplectic integrators, we implement the Leapfrog integrator and the tree-step generalized Nesterov accelerated method (a non-geometric numerical integrator) to simulate the Bregman dynamic in Euclidean case. A conclusion ends this work.

1. Ordinary Differential Equations, Hamiltonian Systems and Geometric Numerical Integrators

Introduction

Differential equations arise in the mathematical formulation of physical phenomena in a wide variety of applications especially in science and engineering. In this chapter, we introduce some basic concepts of Ordinary Differential Equations (ODEs) and study some well known numerical methods for solving ODEs. We also introduce the concept of Hamiltonian systems and sympletic numerical integrators.

The main references for this chapter are Hairer et al. (2006), Leimkuhler and Reich (2004), Mattheij and Molenaar (1996), Ganesh (2007).

1.1 Ordinary Differential Equations (ODEs)

1.1.1 Definitions and properties

In this subsection, we introduce some basic concepts of ordinary differential equations.

Definition 1.1.1. (**Ordinary Differential Equation (ODE)**)

An n^{th} order ODE is an equation involving an unknown function of a single variable (called the independent variable) and its derivatives up to order n. The most general form of an ODE is as follow

$$f(t, y(t), y'(t), \ldots, y^{(n)}(t)) = 0, \tag{1.1.1}$$

where $f\colon I \times \Omega \to \mathbb{R}$ is a continuous function, with Ω a domain of $\mathbb{R}^{n+1}$ and I an interval of $\mathbb{R}$. The natural number n is the order of the ODE.

A function $y : I \to \mathbb{R}$ defined on the interval I is a solution of (1.1.1) if $y \in \mathcal{C}^n(I)$ and $\forall t \in I$, $(t, y(t), y'(t), \ldots, y^{(n)}(t))$ is in the domain of f and $f(t, y(t), y'(t), \ldots, y^{(n)}(t)) = 0$.

An ODE is said to be autonomous if the function f does not depend on t, in this case the equation (1.1.1) can be written as

$$f(y(t),\ y'(t), \ldots, y^{(n)}(t)) = 0.$$

Definition 1.1.2. (**ODE in Normal form**)

An n^{th} order ODE is said to be in normal or explicit form if it can be written as

$$y^{(n)}(t) = f(t, y(t), y'(t), \ldots, y^{(n-1)}(t)),$$

where $f\colon I \times \Omega \to \mathbb{R}$ is a continuous function, with Ω a domain of $\mathbb{R}^n$ and I an interval of $\mathbb{R}$.

Note that, it is not always possible to write an ODE in normal form.

Definition 1.1.3. (**Initial value problem (IVP)**)

Let $\Omega \subset \mathbb{R}^n$ be a domain and $I \subset \mathbb{R}$ an interval, let $t_0 \in I$ and $Y_0 = (y_1, y_2, \ldots, y_n) \in \Omega$ be given. An initial value problem (also called Cauchy problem) for an ODE in normal form is a relation satisfied by an unknown function y given by

$$y^{(n)}(t) = f(t, y(t), y'(t), \ldots, y^{(n-1)}(t)), \quad y^{(i)}(t_0) = y_{i+1},\ i = 0, \ldots, n-1,$$

where $f\colon I \times \Omega$ is a continuous function. The given vector Y_0 is called the initial condition of the IVP.

Definition 1.1.4. (**First-order ODEs**)

An ODE is said to be of first-order if $n = 1$, then the normal form of a first-order ODE is written as

$$y'(t) = f(t, y(t)),$$

where $f\colon I \times \Omega \to \mathbb{R}$ is a continuous function, with Ω a domain of $\mathbb{R}^n$ and I an interval of $\mathbb{R}$. Thus the first order initial value problem is given by the pair of equations

$$y'(t) = f(t, y(t)), \quad y(t_0) = y_0, \text{ where } y_0 \text{ is given.} \tag{1.1.2}$$

Definition 1.1.5. (**First order system of ODE**)

Let $\Omega \subset \mathbb{R}$ be a domain, $I \subset \mathbb{R}$ an interval and p a natural number. A first order system of p ordinary differential equations is given by

$$y'(t) = f(t, y(t)),$$

where $f\colon I \times \Omega \to \mathbb{R}^p$ is a continuous function.

Lemma 1.1.1. Every n^{th} order ODE of the form $y^{(n)}(t) = f(t, y(t), y'(t), \ldots, y^{(n-1)}(t))$ may always be reduced to a system of n first order ODEs.

Indeed, by introducing a transformation $y_1 = y(t),\ y_2 = y'(t), \ldots, y_n = y^{(n-1)}(t)$, we obtain the equivalent system of n first order ODEs

$$\begin{cases} y_1' = y_2 \\ y_2' = y_3 \\ \vdots \\ y_n' = f(t, y_1, y_2, \ldots, y_n). \end{cases}$$

Lemma 1.1.2. Let $\Omega \subset \mathbb{R}^n$ be a domain, $I \subset \mathbb{R}$ an interval and $f\colon I \times \Omega \to \mathbb{R}$ a continuous function. A function Φ defined on I is a solution of the initial value problem $y'(t) = f(t, y(t)),\ y(t_0) = y_0$ if and only if $\Phi(t) \in \Omega,\ \forall\ t \in I$, Φ is continuous and satisfies

$$\Phi(t) = y_0 + \int_{t_0}^{t} f(u, \Phi(u))du.$$

1.1.2 Flow of first order systems of ODEs

Let us consider the following first order system of ODE

$$y'(t) = f(t, y(t)), \quad \text{with} \quad f\colon \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n. \tag{1.1.3}$$

Definition 1.1.6. The flow of (1.1.3) is the map $\varphi\colon \mathbb{R}\times\mathbb{R}^n \to \mathbb{R}^n$ such that for $(t, y_0) \in \mathbb{R}\times\mathbb{R}^n$, $\varphi(t, y_0)$ is the solution of (1.1.3) corresponding to the initial value y_0. Further,

$$\varphi(0, y_0) = y_0 \ \ and \ \ \frac{d}{dt}\varphi(t, y_0) = f(t, \varphi(t, y_0)), \quad \forall (t, y_0) \in \mathbb{R} \times \mathbb{R}^n.$$

In the remaining part of this section, we denote $\varphi(t, y_0)$ by $\varphi_t(y_0)$.

For the given initial state of the problem (1.1.3), the flow φ_t gives the behavior of the system after t units of time. If the problem (1.1.3) has an unique solution, then, the flow is a class $\mathcal{C}^1$ function and satisfies the following properties.

$$\varphi_0 = Id \quad \text{and} \quad \varphi_{s+t} = \varphi_s \circ \varphi_t \tag{1.1.4}$$

and (1.1.4) implies that φ_t is always invertible and $\varphi_t^{-1} = \varphi_{-t}$, φ_t^{-1} is the inverse of φ_t .

At the reverse way, given a (smooth) collection of maps $\varphi_t\colon \mathbb{R}^n \to \mathbb{R}^n$ satisfying $\varphi_0 = Id$ and $\varphi_{s+t} = \varphi_s \circ \varphi_t$, there exist a function f such that φ_t is the flow of the problem $y'(t) = f(t, y(t))$.

Indeed, take $f(t, y(t)) = \dfrac{d\varphi_t}{dt}(y)$.

In general, even if $f(.,.)$ is a continuous function, there is no guarantee that the initial value problem (1.1.2) possesses a unique solution. Fortunately, under a further mild condition on the function $f(.,.)$, the existence and uniqueness of a solution to (1.1.2) can be ensured: the result is encapsulated in the next theorem.

Theorem 1.1.1. *(**Picard's theorem**, Süli (2010))*

Suppose that $f(.,.)$ is a continuous function in some region of $\mathbb{R}^2$ which contains the rectangle

$$R = \{(t, y) :\ t_0 \le t \le T_M,\ \ |y - y_0| \le Y_M\ \}, \tag{1.1.5}$$

where $T_M > 0$, $Y_M > 0$ are constant, and satisfies the Lipschitz condition with constant L, with respect to the second variable, i.e

$$|f(t, y_1) - f(t, y_2)| \le L\,|y_1 - y_2|, \ \textit{for all } (t, y_1) \textit{ and } (t, y_2) \ \textit{ in the rectangle } (1.1.5).$$

Let $M = \max_{(t,y)\in R} |f(t,y)|$.

Suppose also that $M(T_M - t_0) \leq Y_M$*, then there exists a unique continuously differentiable function* $t \longmapsto y(t)$*, defined on some closed interval containing* $[t_0, T_M]$*, which satisfies (1.1.2).*

In most cases, we can prove the existence and uniqueness of our initial value problem but impossible to find that solution at the closed form. Thus we seek the numerical approximation of $y(t)$ on the interval $[t_0,\ T_M]$. In the next section, we look at numerical methods useful for approximating the solution of ODEs.

1.2 Numerical Methods

Let us consider the initial value problem

$$y' = f(t,y), \quad y(t_0) = y_0 \tag{1.2.1}$$

to be solved on the interval $[t_0, T]$, with $t_0,\ T$ real numbers.

Ideally, we divide this interval by the mesh-points $t_n = t_0 + nh, \quad n = 0, \ldots, N,$ where $h = \dfrac{T - t_0}{N}$ is the step size and $N \in \mathbb{N}$. Below, for each n, y_n is a numerical approximation of the exact solution $y(t_n)$ at t_n. Recall that $y(t_0) = y_0$ is given.

1.2.1 Explicit Euler's method

A simple derivation of Euler's method proceeds by first integrating the differential equation between two consecutive mesh-point t_n and t_{n+1}. This results to

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))dt. \tag{1.2.2}$$

Applying the numerical integration rule called the rectangle rule, $\int_{t_n}^{t_{n+1}} f(t,y(t))dt \approx hf(t_n, y(t_n))$, we obtain

$$y(t_0) = y_0,\ y(t_{n+1}) \approx y(t_n) + hf(t_n, y(t_n)).$$

From this, we define the explicit Euler's method by

$$\begin{cases} y_0 = y(t_0) \\ y_{n+1} = y_n + hf(t_n, y_n), \quad n = 0, \ldots, N-1. \end{cases}$$

1.2.2 θ-Method

From (1.2.2), if we replace the rectangle rule by a more general one-parameter family of integration rule of the form

$$\int_{t_n}^{t_{n+1}} f(t, y(t))dt \approx h\left[(1-\theta)f(t_n, y(t_n)) + \theta f(t_{n+1}, y(t_{n+1}))\right], \quad \text{with} \quad \theta \in [0,\ 1],$$

we find that

$$y(t_0) = y_0,\ y(t_{n+1}) \approx y(t_n) + h\left[(1-\theta)f(t_n, y(t_n)) + \theta f(t_{n+1}, y(t_{n+1}))\right].$$

This motivates the introduction of a one-parameter family of methods known as the θ-method

$$\begin{cases} y_0 = y(t_0) \\ y_{n+1} = y_n + h\left[(1-\theta)f(t_n, y_n) + \theta f(t_{n+1}, y_{n+1})\right], \quad \text{with} \quad \theta \in [0,\ 1], \quad n = 0, \ldots, N-1. \end{cases}$$

Many other methods can be derive from the θ-method such as: implicit Euler's method(with $\theta = 1$), trapezoidal rule method(with $\theta = 1/2$) and more.

1.2.3 Implicit Euler's method

We can deduce the implicit Euler's method from the θ-method taking $\theta = 1$. This is defined as

$$\begin{cases} y_0 = y(t_0) \\ y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}), \quad n = 0, \ldots, N-1. \end{cases}$$

1.2.4 Trapezoidal rule method

This method is deduced from the θ-method by taking $\theta = 1/2$. It is defined as

$$\begin{cases} y_0 = y(t_0) \\ y_{n+1} = y_n + \dfrac{h}{2}\left[f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right], \quad n = 0, \ldots, N-1. \end{cases}$$

Remark 1. The θ-method is an explicit method for $\theta = 0$, and an implicit if $0 < \theta \leq 1$.

1.2.5 General One-step methods

Definition 1.2.1. A numerical method $(y_n,\ n = 0, \ldots, N)$ is said to be a One-step method if it can be written as

$$\begin{cases} y_0 = y(t_0) \\ y_{n+1} = y_n + h\Phi(t_n, y_n; h), \quad n = 0, \ldots, N-1, \end{cases} \tag{1.2.3}$$

where $\Phi(.,.;.)$ is a continuous function of its variables.

In order to assess the accuracy of a numerical method, we define the global error e_n at each mesh-point t_n as $e_n = y(t_n) - y_n$ and the truncation error T_n as

$$T_n = \frac{y(t_{n+1}) - y(t_n)}{h} - \Phi(t_n, y_n; h).$$

Definition 1.2.2. The numerical method (1.2.3) is consistent with the initial value problem (1.2.1) if the truncation error is such that, for any $\epsilon > 0$ there exists a positive constant $h(\epsilon)$ for which $|T_n| < \epsilon$ for $0 < h < h(\epsilon)$ and any pair of points $(t_n, y(t_n)),\ (t_{n+1}, y(t_{n+1}))$ on any solution curve in R.

Theorem 1.2.1. *Suppose that the solution of the initial value problem (1.2.1) lies in R (R is the rectangle defined in (1.1.5) as does its approximation generated from (1.2.3) when $h < h_0$ (h_0 is a given positive constant). Suppose also that the function $\Phi(.,.;.)$ is uniformly continuous on $R \times [0,\ h_0]$ and satisfies the consistency condition and the Lipschitz condition with respect to the second variable with constant L_Φ, i.e*

$$|\Phi(t, y_1; h) - \Phi(t, y_2; h)| \leq L_\Phi |y_1 - y_2|, \quad \textit{for all} \quad (t, y_1; h), (t, y_2; h) \in R \times [0,\ h_0].$$

Then if successive approximation sequence (y_n), generated for $t_n = t_0 + nh,\ n = 0, \ldots, N$ are obtained from (1.2.3) with successively smaller values of h each less than h_0, then the numerical solution of the initial value problem (1.2.1) converges to the exact solution as $h \to 0$ in the sense that $|y(t_n) - y_n| \to 0 \quad as \quad h \to 0$.

Definition 1.2.3. A numerical method is said to have order of accuracy p, if p is the largest positive integer such that, for any sufficiently smooth solution curve in R of the initial value problem (1.1.2), there exist constants K and h_0 such that

$$|T_n| \leq Kh^p \ \ for \ \ 0 < h < h_0,$$

for any pair of points $(t_n, y(t_n)),\ \ (t_{n+1}, y(t_{n+1}))$ on the solution curve in R.

We say that a one-step method (1.2.3) has the order of convergence p, if the local error (for all nice enough problems (1.2.1)) verifies $y(t_0 + h) - y_1 = O(h^{p+1})$. Recall that $g(h) = O(h^{p+1}), h \to 0$ if and only if there exists a positive constant C such that $|g(h)| \leq Ch^{p+1}$ for all h small enough.

1.2.6 Runge-Kutta methods

Definition 1.2.4. Let $s > 0$ be an integer, $b_i, a_{ij} \in \mathbb{R}$, for $i, j = 1, \ldots s,\ c_1 = 0$ and $c_i := \sum_{j=1}^{i-1} a_{ij}$, for $i \geq 2$. A Runge-Kutta method is a one-step numerical method defined as follow

$$\begin{cases} k_1 = f(t_0, y_0) \\ k_2 = f(t_0 + c_2 h, y_0 + h a_{21} k_1) \\ \vdots \\ k_s = f(t_n + c_s h,\ y_n + h \sum_{j=1}^{s} a_{sj} k_j) \\ y_{n+1} = y_n + h \sum_{j=1}^{s} b_j k_j. \end{cases}$$

The Runge-Kutta scheme can be summarized in a table known as Butcher tableau. If $a_{ij} = 0$

c_1	a_{11}	a_{12}	a_{13}	$\dots$	a_{1s}
c_2	a_{21}	a_{22}	a_{23}	$\dots$	a_{2s}
c_3	a_{31}	a_{32}	a_{33}	$\dots$	a_{3s}
$\vdots$	$\dots$	$\dots$	$\dots$	$\ddots$	$\dots$
c_s	a_{s1}	a_{s2}	$\dots$	$a_{s,s-1}$	a_{ss}
	b_1	b_2	$\dots$	b_{s-1}	b_s

Table 1.1: The Butcher tableau of a Runge-Kutta scheme.

for $i \leq j$, we have an explicit one-step Runge-Kutta method. In this case, the Butcher tableau is as shown in Table 1.2.

c_1=0	0	0	0	$\dots$	0
c_2	a_{21}	0	0	$\dots$	0
c_3	a_{31}	a_{32}	0	$\dots$	0
$\vdots$	$\dots$	$\dots$	$\dots$	$\ddots$	0
c_s	a_{s1}	a_{s2}	$\dots$	$a_{s,s-1}$	0
	b_1	b_2	$\dots$	b_{s-1}	b_s

Table 1.2: Butcher tableau for an explicit Runge-Kutta scheme method.

Example 1.2.1. We consider some particular cases of Runge-Kutta methods and present their Butcher tableaux.

1. Runge-Kutta of order 1.

$$y_{t_{n+1}} = y_{t_n} + hf(t_n, y_{t_n})$$

0	0
	1

(a) RK1 (Euler scheme)

2. Runge-Kutta of order 2

$$\begin{cases} k_1 = f(y_n, t_n) \\ k_2 = f(y_n + hk_1, t_n + h) \\ y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2) \end{cases}$$

0	0	0
1	1	0
	$1/2$	$1/2$

(b) RK2

2. Runge-Kutta of order 3.

Runge-Kutta of order 4.

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 = f(t_n + h, y_n + 2hk_2 - hk_1) \\ y_{n+1} = y_n + \frac{h}{6}(k_1 + 4k_2 + k_3) \end{cases}$$

0	0	0	0
1/2	1/2	0	0
1	-1	2	0
	1/6	2/3	1/6

(a) RK3

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 = f(t_n + h, y_n + hk_3) \\ y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases}$$

0	0	0	0	0
1/2	1/2	0	0	0
1/2	0	1/2	0	0
1	0	0	1	0
	1/6	2/6	2/6	1/6

(b) RK4

1.3 Hamiltonian Systems

Many important problems in mechanic and machine learning can be described by Hamiltonian systems. This section presents the definition of Hamiltonian systems and the study some of their properties. Denote $\dot{u} := \dfrac{du}{dt}$ for a given function u.

1.3.1 Definition and examples

Definition 1.3.1. Let $d > 0$ be an integer and $H : \mathbb{R}^d \times \mathbb{R}^d \longmapsto \mathbb{R}$ be a given sufficiently differentiable function. A Hamiltonian system consists of the following set of differential equations

$$\dot{p}_i = -\frac{\partial H}{\partial q_i}(p, q), \quad \dot{q}_i = \frac{\partial H}{\partial p_i}(p, q), \quad i = 1, \dots d$$

or using a more compact notation

$$\dot{p} = -\nabla_q H(p, q) = -\left(\frac{\partial H}{\partial q}\right)^T, \quad \dot{q} = \nabla_p H(p, q) = -\left(\frac{\partial H}{\partial p}\right)^T$$

or even more compact

$$\dot{y} = J^{-1}\nabla H(y), \quad y = (p, q)^T, \quad J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}, \tag{1.3.1}$$

where I is the identity matrix in $\mathbb{R}^{d\times d}$.

In the example of mechanical systems, one can think of the Hamiltonian as the total energy of the system, with $q = (q_1, \ldots, q_d)^T$ as general position coordinates (this can be for examples, Cartesian coordinate, angle, arc lengths along a curve, etc.), $p = (p_1, \ldots, p_d)^T$ the momentum.

Example 1.3.1. (**Mathematical pendulum**)

The mathematical pendulum (mass $m = 1$, massless rod of length $l = 1$, gravitational acceleration $g = 1$) is presented in Figure 1.1. This is an example of Hamiltonian system with one degree of freedom. We have $H(p, q) = T(p, q) + V(p, q)$, where $T(p, q) = \dfrac{p^2}{2m}, m = 1$, and

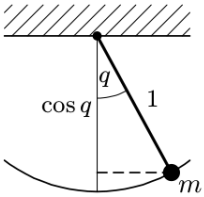


Figure 1.1: Mathematical pendulum (Hairer et al. (2006)).

$V(q) = -cos(q)$.

Then

$$H(p, q) = \frac{p^2}{2} - cos(q)$$

so that, the equation of motion in this case is defined as

$$\dot{p} = -sin(q), \quad \dot{q} = p.$$

1.3.2 Geometric properties of Hamiltonian systems

In this subsection, we will consider some important properties of Hamiltonian systems including **conservation of the energy, symplecticity** and **first integrals**.

Definition 1.3.2. A non-constant function $I(y)$ is a first integral of the problem $y' = f(y)$ if $I'(y)f(y) = 0$, for all y. This implies that every solution $y(t)$ of $y' = f(y)$, satisfies $I(y(t)) = constant$.

1.3.2.1 Property 1 : Conservation of the total energy

Conservation of total energy of Hamiltonian systems (1.3.1) imply that

$$H(p(t), q(t)) = H(p(0), q(0)) \text{ for all time } t$$

along the exact solution of the problem. Indeed, one has

$$\begin{aligned}\frac{d}{dt}H(p(t),q(t)) &= \frac{\partial H}{\partial p}\frac{dp}{dt}+\frac{\partial H}{\partial q}\frac{dq}{dt}\\ &= \frac{\partial H}{\partial p}\dot{p}+\frac{\partial H}{\partial p}\dot{q}\\ &= \frac{\partial H}{\partial p}\left(-\frac{\partial H}{\partial q}\right)^T+\frac{\partial H}{\partial q}\left(-\frac{\partial H}{\partial p}\right)^T\\ &= 0.\end{aligned}$$

Thus, the total energy along the exact solution of Hamiltonian systems is conserved.

1.3.2.2 Property 2 : Symplecticity

One of the most relevant properties of Hamiltonian systems is symplecticity. The basic objects are two-dimensional parallelogram lying in $\mathbb{R}^d$.

Let us consider the parallelogram $\mathcal{P}\subset\mathbb{R}^{2d}$ spanned by the vector $\xi=\begin{pmatrix}\xi^p\\ \xi^q\end{pmatrix}$ and $\eta=\begin{pmatrix}\eta^p\\ \eta^q\end{pmatrix}$. Further, the application (bilinear form)

$$\begin{aligned}&\omega\colon \mathbb{R}^{2d}\times\mathbb{R}^{2d}\to\mathbb{R}\\ &(\xi,\eta)\mapsto \xi^T J\eta. \qquad with\ \ J=\begin{pmatrix}0 & I\\ -I & 0\end{pmatrix}.\end{aligned}$$

Example 1.3.2. In the case $d=1$, one has

$$\omega(\xi,\eta)=\xi^p\eta^q-\xi^q\eta^p=det\begin{pmatrix}\xi^p & \eta^p\\ \xi^q & \eta^q\end{pmatrix}.$$

This is the oriented area of the parallelogram $\mathcal{P}$. In the case $d>1$, we consider the sum of oriented areas of projections of $\mathcal{P}$ onto the coordinate plane (p_i,q_i):

$$\begin{aligned}\omega(\xi,\eta)=\xi^T J\eta &= \sum_{i=1}^{n} det\begin{pmatrix}\xi_i^p & \eta_i^p\\ \xi_i^q & \eta_i^q\end{pmatrix}\\ &= \sum_{i=1}^{n}\left(\xi_i^p\eta_i^q-\xi_i^q\eta_i^p\right).\end{aligned}$$

Definition 1.3.3. A linear map $A\colon \mathbb{R}^{2d}\times\mathbb{R}^{2d}\to\mathbb{R}$ is said to be symplectic if $\omega(A\xi,A\eta)=\omega(\xi,\eta)$ for all $\xi,\ \eta\in\mathbb{R}^{2d}$ or equivalently, if $A^TJA=J$ with $\omega(.)$ and J defined above.

In general $(n>1)$, symplecticity means that, the sum of the oriented areas of the projection of $\mathcal{P}$ onto (p_i,q_i) is the same as that for the transformed parallelograms $A(\mathcal{P})$.

Definition 1.3.4. (**Symplectic map**) A differentiable map $g\colon \Omega\subset\mathbb{R}^{2n}\to\mathbb{R}^{2n}$ is said to be symplectic if the Jacobian matrix $g'(p,q)$ is everywhere symplectic (according to the above definition). This means that,

$(g'(p,q))^TJg'(p,q)=J$ or $\omega(g'(p,q)\xi,g'(p,q)\eta)=\omega(\xi,\eta)$ for all $p,q\in\mathbb{R}^n$.

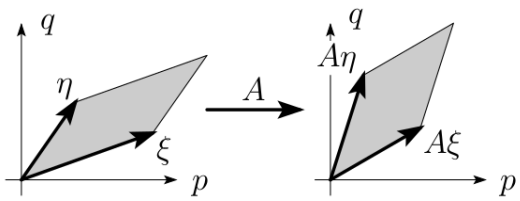


Figure 1.2: Symplecticity (area preservation) of a linear map (Hairer et al. (2006)).

The most important characteristics of Hamiltonian systems is given by the following theorems.

Theorem 1.3.1. (Poincaré (1899)) *For every fixed time t, the flow φ_t (i.e. the mapping that advances the solution by time t : $\varphi_t(p_0, q_0) = (p(t, p_0, q_0), q(t, p_0, q_0))$) of the Hamiltonian system (1.3.1) is a symplectic transformation.*

Proof. Let $y_0 = (p_0, q_0)$. The derivative $\dfrac{\partial \varphi_t}{\partial y_0}$ is the solution of the variational equation which, for the Hamiltonian system (1.3.1), is of the form $\dot{\psi} = J^{-1}\nabla^2 H(\varphi_t(y_0))\psi$, where $\nabla^2 H(p, q)$ is the Hessian matrix of $H(p, q)$. Therefore we get

$$\begin{aligned}
\frac{d}{dt}\left[\left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right)\right] &= \left(\frac{\partial \varphi_t}{\partial y_0}\right)'^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right) + \left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right)' \\
&= \left[J^{-1}\nabla^2 H(\varphi_t(y_0))\frac{\partial \varphi_t}{\partial y_0}\right]^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right) + \left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left[J^{-1}\nabla^2 H(\varphi_t(y_0))\frac{\partial \varphi_t}{\partial y_0}\right] \\
&= \left(\frac{\partial \varphi_t}{\partial y_0}\right)^T \nabla^2 H(\varphi_t(y_0)) J^{-T} J \left(\frac{\partial \varphi_t}{\partial y_0}\right) + \left(\frac{\partial \varphi_t}{\partial y_0}\right)^T \nabla^2 H(\varphi_t(y_0)) \left(\frac{\partial \varphi_t}{\partial y_0}\right).
\end{aligned}$$

(Hessian matrix symmetric property)

Since $J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}$, then $J^T = -J$ hence $J^{-T}J = -I$.

Then we obtain

$$\frac{d}{dt}\left[\left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right)\right] = 0.$$

Hence, $\left(\dfrac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\dfrac{\partial \varphi_t}{\partial y_0}\right)$ is constant for all time t.

Then

$$\begin{aligned}
\left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right) &= \left(\frac{\partial \varphi_0}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_0}{\partial y_0}\right) \\
&= IJI \\
&= J.
\end{aligned}$$

Hence, φ_t is a symplectic transformation. □

This result is illustrated below in Figure 1.3 with the pendulum problem using the normalization ($m = l = g = 1$). The level curves of the Hamiltonian function $H(p,q) = \frac{p^2}{2} - cos(q)$, are displayed together with illustrations of the area preservation of the flow φ_t, with different values of t.

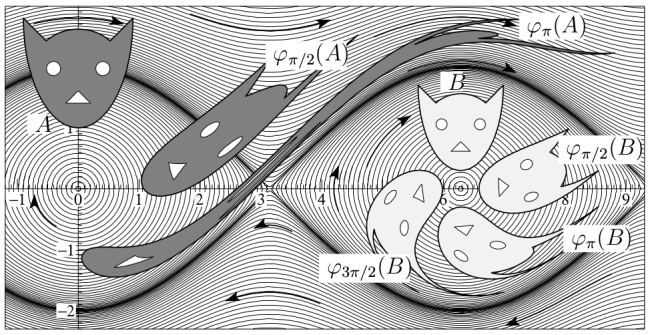


Figure 1.3: Area preservation of the flow of Hamiltonian systems (Hairer et al. (2006)).

Theorem 1.3.2. *The flow φ_t of a differential equation $y' = f(y)$ is a symplectic transformation for all time t if and only if locally*

$$f(y) = J^{-1}\nabla H(y), \text{ for some function } \quad H(y).$$

This result shows that symplecticity of the flow is a characteristic property of Hamiltonian systems.

Proof. The necessity follows from the above Poincaré's theorem.

Let us assume that the flow φ_t is symplectic, we have to prove the existence of a function $H(y)$ such that $f(y) = J^{-1}\nabla H(y)$ locally.

From the definition (1.3.4) we have

$$\left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right) = J \quad (constant)$$

then

$$\frac{d}{dt}\left[\left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right)\right] = 0$$

otherwise, using the fact that $\dfrac{\partial \varphi_t}{\partial y_0}$ is the solution of $\dot{\psi} = J^{-1}\nabla^2 H(\varphi_t(y_0))\psi$, we have

$$\frac{d}{dt}\left[\left(\frac{\partial \varphi_t}{\partial y_0}\right)^T J \left(\frac{\partial \varphi_t}{\partial y_0}\right)\right] = \left(\frac{\partial \varphi_t}{\partial y_0}\right)\left(f'(\varphi_t(y_0))^T J + J f'(\varphi_t(y_0))\right)\left(\frac{\partial \varphi_t}{\partial y_0}\right).$$

So

$$\left(\frac{\partial \varphi_t}{\partial y_0}\right)\left(f'(\varphi_t(y_0))^T J + J f'(\varphi_t(y_0))\right)\left(\frac{\partial \varphi_t}{\partial y_0}\right) = 0,$$

taking $t = 0$, we get

$$f'(y_0)^T J + J f'(y_0) = 0$$

i.e

$$-f'(y_0)^T J^T + J f'(y_0) = 0 \qquad (J = -J^T)$$

i.e

$$(J f'(y_0))^T = J f'(y_0).$$

So $Jf'(y_0)$ is a symmetric matrix for all y_0, by the integrability lemma (given below), there exists some function $H(y)$ such that

$$Jf'(y) = \nabla H(y)$$

i.e

$$f'(y) = J^{-1} \nabla H(y).$$

□

Lemma 1.3.1. (**Integrability Lemma**)

Let $\Omega \subset \mathbb{R}^n$ be an open set and $f : \Omega \to \mathbb{R}^n$ be a continuously differentiable function. Assume that the Jacobian $f'(y)$ is symmetric for all $y \in \Omega$, then for every $y_0 \in \Omega$, there exists a neighborhood of y_0 and a function $H(y)$ such that $f(y) = \nabla H(y)$ on this neighborhood.

Indeed, assume that $y_0 = 0$, and consider a ball around y_0 which is contained in Ω. On this ball, we set

$$H(y) = \int_0^1 y^T f(ty) dt + \text{Constant}.$$

Differentiating with respect to y_k, and using the symmetry assumption $\dfrac{\partial f_i}{\partial y_k} = \dfrac{\partial f_k}{\partial y_i}$, we have

$$\frac{\partial H}{\partial y_k} = \int_0^1 (f_k(ty) + y^T \frac{\partial f}{\partial y_k}(ty)t) dt = \int_0^1 \frac{d}{dt}(t f_k(ty)) dt = f_k(y).$$

Thus $f(y) = \nabla H(y)$.

1.3.3 Generating functions

Like Hamiltonian system, symplectic maps can be described by only one scalar function called generating function. The following results are conveniently formulated in the notation of differential forms. For a function $f(y)$, denoted by $df = df(y)$, its (Fréchet) derivative is the linear map

$$df(y)(\xi) = f'(y)(\xi) = \sum_{i=1}^{n} \frac{\partial f}{\partial y_i}(y)(\xi).$$

For the special case $f(y) = y_k$, we denote the derivative as dy_k and $dy_k(\xi) = \xi_k$ as the projection onto the k^{th} component. With $dy = (dy_1, \ldots, dy_n)^T$, we have $df = \sum\limits_{i=i}^{n} \dfrac{\partial f}{\partial y_i}(y)\xi_i$.

For a function $S(p,q)$, we use the notation

$$\begin{aligned} dS(p,q) &= S_p dq + S_q dp \\ &= \sum_{i=1}^{d} \left(\frac{\partial S}{\partial p_i}(p,q) dp_i + \frac{\partial S}{\partial q_i} dq_i \right). \end{aligned}$$

Theorem 1.3.3. *A map $\varphi\colon (p,q) \longmapsto (P,Q)$ is symplectic if and only if there exist locally a function $S(p,q)$ such that*

$$P^T dQ - p^T dq = dS.$$

This means that $P^T dQ - p^T dq$ is a total differential.

Theorem 1.3.4. *Let $\varphi\colon (p,q) \longmapsto (P,Q)$ be a smooth transformation close to the identity function. It is symplectic if and only if one of the following conditions holds locally:*

1) $Q^T dP + p^T dq = d(P^T q + S^1)$ for some function $S^1(P,q)$;

2) $P^T dQ + q^T dp = d(p^T Q - S^2)$ for some function $S^2(p,Q)$;

3) $(Q-q)^T d(P+p) - (P-p)^T d(Q+q) = 2dS^3$ for some function $S^3((P+p)/2, (Q+q)/2)$.

For the proofs of these two theorems, see Hairer et al. (2006).

1.4 First Geometric Numerical Integrators

All the characteristic properties of Hamiltonian systems enumerated above have motivated the search for numerical integrators that preserve them, and more specifically its symplectic character. This section is devoted to the notion of symplectic integrators and some examples of symplectic numerical methods.

1.4.1 Definition and examples

Definition 1.4.1. A numerical one-step method $y_{n+1} = \Phi_h(y_n)$ is said to be symplectic if, when applied to a Hamiltonian system, the discrete flow $y \to \Phi_h(y)$ is a symplectic transformation for all sufficiently small step sizes.

Let us consider the following Hamiltonian system in the variable $y = (p,\ q)$

$$\begin{aligned} \dot{p} &= -\nabla_q H(p,q) \\ \dot{q} &= \nabla_p H(p,q) \end{aligned} \qquad \text{or} \quad \text{equivalently} \quad \dot{y} = J^{-1} \nabla H(y).$$

Below, we present some symplectic methods.

Example 1.4.1. (Symplectic Euler method, de Vogelaere 1956)

The so-called symplectic Euler methods

$$\begin{cases} p_{n+1} = p_n - h\nabla_q H(p_{n+1}, q_n) \\ q_{n+1} = q_n + h\nabla_p H(p_{n+1}, q_n) \end{cases} \quad \text{or} \quad \begin{cases} p_{n+1} = p_n - h\nabla_q H(p_n, q_{n+1}) \\ q_{n+1} = q_n + h\nabla_p H(p_n, q_{n+1}) \end{cases}$$

and are symplectic methods of order 1.

We can see it as a consequence of the first two condition of theorem 1.3.4. We can also remark that, this method is a combination of the explicit and implicit Euler's method.

Example 1.4.2. (Störmer-Verlet method)

The Störmer-Verlet method

$$\begin{cases} p_{n+1/2} = p_n - \dfrac{h}{2}\nabla_q H(p_{n+1/2}, q_n) \\ q_{n+1} = q_n + \dfrac{h}{2}\left(\nabla_p H(p_{n+1/2}, q_n) + \nabla_p H(p_{n+1/2}, q_{n+1})\right) \\ p_{n+1} = p_{n+1/2} - \dfrac{h}{2}\nabla_q H(p_{n+1/2}, q_{n+1}) \end{cases}$$

or

$$\begin{cases} q_{n+1/2} = q_n + \dfrac{h}{2}\nabla_p H(p_n, q_{n+1/2}) \\ p_{n+1} = p_n - \dfrac{h}{2}\left(\nabla_q H(p_{n+}, q_{n+1/2}) + \nabla_q H(p_{n+1}, q_{n+1/2})\right) \\ q_{n+1} = q_{n+1/2} + \dfrac{h}{2}\nabla_p H(p_{n+1}, q_{n+1/2}) \end{cases}$$

are symplectic methods of order 2.

We can remark that, those numerical methods are a composition of two symplectic Euler's method with step size $h/2$(see below for more details).

For a second order differential equation $\ddot{q} = -\nabla U(q)$ for which the Hamiltonian is $H(p, q) = \dfrac{1}{2}p^T q + U(q)$, the Stömer-Varlet method becomes

$$q_{n+1} - 2q_n + q_{n+1} = -h^2\nabla U(q_n), \quad p_n = \frac{q_{n+1} - q_n}{2h}.$$

Example 1.4.3. (Implicit midpoint)

The implicit midpoints rule

$$y_{n+1} = y_n + hJ^{-1}\nabla H(\frac{y_{n+1} + y_n}{2})$$

is symplectic of order 2. The symplecticity is a consequence of the third characterization of theorem 1.3.2 and of order 2 comes from the symmetry of the method.

1.4.2 Symplectic Runge-Kutta methods

Recall that, the family of Runge- Kutta method is family of the numerical methods defined as

$$\begin{cases} k_1 = f(t_0, y_0) \\ k_2 = f(t_0 + c_2 h, y_0 + h a_{21} k_1) \\ \vdots \\ k_s = f(t_n + c_s h,\ y_n + h \sum_{j=1}^{s} a_{sj} k_j) \\ y_{n+1} = y_n + h \sum_{j=1}^{s} b_j k_j, \end{cases} \tag{1.4.1}$$

where $s > 0$ is an integer, $b_i, a_{ij} \in \mathbb{R}$ for $i, j = 1, ..., s,\quad c_1 = 0$ and $c_i := \sum_{j=1}^{i-1} a_{ij}$, for $i \geq 2$.

Theorem 1.4.1. *If a Runge-Kutta method conserves quadratic first integral i.e $I(y_1) = I(y_0)$ then it is symplectic. Here we recall that $I(y) = y^T C y$ is a first integral of $\dot{y} = f(y)$ and C symmetric matrix.*

Proof. See (Hairer et al. (2006)) □

Definition 1.4.2. Let $c_1,\ c_2, \ldots, c_s$ be distinct real numbers (usually $0 \leq c_i \leq 1$). The Collocation polynomial $u(t)$ is a polynomial of degree s satisfying

$$\begin{cases} u(t_0) = y_0 \\ u'(t_0 + c_i h) = f(t_0 + c_i h,\ u(t_0 + c_i h)), \quad i = 1, \ldots, s \end{cases} \tag{1.4.2}$$

and the numerical solution of the collocation method is defined by $y_1 = u(t_0 + h)$.

For $s = 1$, the polynomial has the form $u(t) = y_0 + (t - t_0)k,$ with $k = f(t_0 + c_1 h,\ y_0 + h c_1)$. We observe that the explicit and implicit Euler methods and midpoint rule are collocation method with $c_1 = 0,\ c_1 = 1,\ c_1 = 1/2$ respectively.

Theorem 1.4.2. *(Guillou & Soulé 1969, Wright 1990)*

The collocation method of above definition is equivalent to the s-stage Runge-Kutta method with coefficients

$$a_{ij} = \int_0^{c_i} l_j(\xi) d\xi, \quad b_i = \int_0^1 l_i(\xi) d\xi,$$

where $l_i(\xi)$ is the Lagrange polynomial

$$l_i(\xi) = \prod_{j \neq i} \frac{\xi - c_j}{c_i - c_j}.$$

Proof. see Hairer et al. (2006) □

Theorem 1.4.3. *The Gauss collocation methods conserve quadratic first integrals an hence are symplectic by Theorem 1.4.1.*

Proof. Let $u(t)$ be the Gauss collocation polynomial of the Gauss collocation method, assume that $I(y) = y^T C y$, with a symmetric matrix C, is a first integral of $y' = f(y)$. Since $\frac{d}{dt} I(u(t)) = 2u(t)Cu'(t)$, it follows from $u(t_0) = y_0$ and $u(t_0 + h) = y_1$ that

$$\int_{t_0}^{t_0+h} \frac{d}{dt} I(u(t)) = 2 \int_{t_0}^{t_0+h} u(t)^T C u'(t) dt$$

i.e $I(u(t_0 + h)) + I(u(t_0)) = 2 \int_{t_0}^{t_0+h} u(t)^T C u'(t) dt.$

So $I(y_1) - I(y_0) = 2 \int_{t_0}^{t_0+h} u(t)^T C u'(t) dt.$

Hence $y_1^T C y_1 - y_0^T C y_0 = 2 \int_{t_0}^{t_0+h} u(t)^T C u'(t) dt.$

Using the relation $u(t_0 + c_i h)^T C u'(t_0 + c_i h) = u(t_0 + c_i h)^T C f(u(t_0 + c_i h)) = 0,$

we get $2 \int_{t_0}^{t_0+h} u(t)^T C u'(t) dt = 0.$

Hence $y_1^T C y_1 = y_0^T C y_0.$

□

Theorem 1.4.4. *If the coefficients of the Runge-Kutta (1.4.1) method satisfy*

$$b_i a_{ij} + b_j a_{ji} = b_i b_j \quad for \quad all \quad i, j = 1, \ldots, s \tag{1.4.3}$$

then it conserves quadratic first integrals. Hence, such numerical methods are symplectic.

Proof. From the definition of a Runge-Kutta method, we have

$$y_1 = y_0 + h \sum_{j=1}^{s} b_j k_j \quad then \quad y_1^T = y_0^T + h \sum_{j=1}^{s} b_j k_j^T.$$

Thus,

$$y_1^T C y_1 = y_0^T C y_0 + h \sum_{j=1}^{s} b_j y_0^T C k_j + h \sum_{i=1}^{s} b_i k_i^T C y_0 + h^2 \sum_{i,j=1}^{s} b_i b_j k_i^T C k_j,$$

letting $k_i = f(Y_i)$ with $Y_i = y_0 + h \sum_{j=1}^{s} a_{ij} k_j$, we have

$$y_1^T C y_1 = y_0^T C y_0 + 2h \sum_{i=1}^{s} b_i Y_i^T C f(Y_i) + h \sum_{i=1}^{s} b_i k_i^T C y_0 + h^2 \sum_{i,j=1}^{s} (b_i b_j - b_i a_{ij} - b_j a_{ij}) k_i^T C k_j.$$

Using the assumption $y^T C f(y) = 0$ and (1.4.3), we get $y_1^T C y_1 = y_0^T C y_0$. □

1.4.3 Adjoint and symmetric methods

The flow φ_t of an autonomous differential equation $y' = f(y)$ satisfies $\varphi_{-t}^{-1} = \varphi_t$. Note that, this property is not generally shared by the one-step map Φ_h of a numerical method.

Definition 1.4.3. The adjoint method Φ_h^* of a method Φ_h is the inverse map of the original method with reversed time step $-h$, i.e.

$$\Phi_h^* := \Phi_{-h}^{-1}.$$

In other words, $y_1 = \Phi_h^*(y_0)$ is implicitly defined as the relation $\Phi_{-h}(y_1) = y_0$. A method for which $\Phi_h^* = \Phi_h$ is called symmetric.

According to this definition, the scheme $y_1 = \Phi_h(y_0)$ is symmetric if and only if exchanging $h \leftrightarrow -h$ and $y_0 \leftrightarrow y_1$, we achieve the same expression i.e $\Phi_{-h}(y_1) = y_0$. Notice that the adjoint method satisfies the usual properties such as

$$(\Phi_h^*)^* = \Phi \quad \text{and} \quad (\Phi_h \circ \Psi_h)^* = \Psi_h^* \circ \Phi_h^*, \quad \text{for any one-step methods } \Phi_h \text{ and } \Psi_h.$$

Theorem 1.4.5. *Let φ_t be the exact flow of $y' = f(y)$ and let Φ_h be a one-step method of order r satisfying*

$$\Phi_h(y_0) = \varPhi_h(y_0) + C(y_0)h^{r+1} + O(h^{r+2}).$$

The adjoint method Φ_h^* has the same order r and satisfies

$$\Phi_h^*(y_0) = \varPhi_h(y_0) + (-1)^r C(y_0)h^{r+1} + O(h^{r+2}).$$

If the method is symmetric, its maximal order is even.

Proof. See (Hairer et al. (2006)). □

1.4.4 Composition methods

Let Φ_h be a basic numerical method and $\gamma_1, \ldots, \gamma_s$ real numbers. We call its composition with step sizes $\gamma_1 h, \ldots, \gamma_s h$, the composition method defined as

$$\Psi_h = \Phi_{\gamma_1 h} \circ \Phi_{\gamma_2 h} \circ \cdots \circ \Phi_{\gamma_s h}.$$

The aim is to increase the order while preserving desirable properties like symplecticity of the basic numerical method.

Theorem 1.4.6. *Let Φ_h be a one-step method of order p, if $\gamma_1 + \cdots + \gamma_s = 1$ and $\gamma_1^{p+1} + \cdots + \gamma_s^{p+1} = 0$. Then the composition method is at least of order $p + 1$.*

Proof. See Hairer et al. (2006) □

A more general composition method including the adjoint method is given by the following formula

$$\Psi_h = \Phi_{\alpha_s h} \circ \Phi^*_{\beta_s h} \circ \cdots \circ \Phi^*_{\beta_2 h} \circ \Phi_{\alpha_1 h} \circ \Phi^*_{\beta_1 h}$$
$$\text{with} \quad \beta_1 + \alpha_1 + \cdots + \beta_s + \alpha_s = 1 \quad \text{and} \quad (-1)\beta_1^{r+1} + \alpha_1^{r+1} + \cdots + (-1)\beta_s^{r+1} + \alpha_s^{r+1} = 0.$$

Composition methods allow to increase odd r order as well. For example, the composition of every consistent one-step method Φ_h of order 1 gives the second-order symmetric method defined as

$$\Psi_h = \Phi_{\alpha_s h/2} \circ \Phi^*_{\beta_s h/2}. \tag{1.4.4}$$

We can see for example, if $\Phi_{\alpha_s h}$ is the explicit (resp. implicit) Euler method, then Ψ_h in (1.4.4) becomes the implicit midpoint (resp. trapezoidal) rule. If $\Phi_{\alpha_s h}$ is the symplectic Euler method, then the composed method Ψ_h in (1.4.4) is the Störmer–Verlet method.

1.4.5 Splitting methods

The main idea of splitting method is as follow. Let us consider an arbitrary system

$$y' = f(y) \quad \text{in} \quad \mathbb{R}^n.$$

Suppose that the vector field is "split" as

$$y' = f^{[1]}(y) + f^{[2]}(y), \tag{1.4.5}$$

and the exact flow $\varphi^{[1]}$ and $\varphi^{[2]}$ of the systems $y' = f^{[1]}(y)$ and $y' = f^{[2]}(y)$ respectively, can be calculated explicitly, then we can compose them to get the numerical approximation y of (1.4.5).

1.4.5.1 Lie-Trotter Splitting

From a given initial value y_0, we first solve the first system to obtain a value $y_{1/2}$. From this value, we integrate the second system to obtain y_1 and get the numerical integrators

$$\Phi_h = \varphi_h^{[1]} \circ \varphi_h^{[2]} \quad \text{and} \quad \Phi_h^* = \varphi_h^{[2]} \circ \varphi_h^{[1]}, \tag{1.4.6}$$

were one is the adjoint of the other. By Taylor expansion, we get

$(\varphi_h^{[1]} \circ \varphi_h^{[2]}(y_0) = \varphi_h(y_0) + O(h^2))$, so that both methods give an approximation of order 1 to the solution of (1.4.5).

1.4.5.2 Strang splitting

Another idea is to use a symmetric version of the above by letting

$$\Phi_h^{[S]} = \varphi_{h/2}^{[1]} \circ \varphi_h^{[2]} \circ \varphi_{h/2}^{[1]}. \tag{1.4.7}$$

By breaking up in (1.4.7) $\varphi_h^{[2]} = \varphi_{h/2}^{[2]} \circ \varphi_{h/2}^{[2]}$, we see that the Strang splitting

$\Phi_h^{[S]} = \Phi_{h/2} \circ \Phi^*_{h/2}$ is the composition of the Lie-Trotter method (1.4.6) and its adjoint with halved step size. Thus the Strang splitting formula is symmetric and of order 2.

1.4.5.3 General splitting procedure

In a similar way to the general idea of composition method (1.4.4), we can form with arbitrary coefficients $a_1, b_1, \ldots, a_m, b_m$ (where eventually a_1 *or* b_m or both are null)

$$\Psi_h = \varphi^{[2]}_{b_m h} \circ \varphi^{[1]}_{a_m h} \circ \varphi^{[2]}_{b_{m-1} h} \circ \cdots \circ \varphi^{[1]}_{a_2 h} \circ \varphi^{[2]}_{b_1 h} \circ \varphi^{[1]}_{a_1 h}, \tag{1.4.8}$$

and try to increase the order of the scheme by suitably determining the free coefficients.

A close connection between the theories of splitting method (1.4.8) and composition method (1.4.4) was discovered by McLachlan (1995). Indeed, if we put $\beta_1 = a_1$ and break up $\varphi^{[2]}_{b_1 h} = \varphi^{[2]}_{\alpha_1 h} \circ \varphi^{[2]}_{\beta_1 h}$ where α_1 is given below, further $\varphi^{[1]}_{a_2 h} = \varphi^{[1]}_{\beta_2 h} \circ \varphi^{[1]}_{\alpha_1 h}$ and so on. We see from (1.4.6) that, Ψ_h defined in (1.4.4) is identical with that defined in (1.4.8),

$$\text{where} \quad \Phi_h = \varphi^{[1]}_h \circ \varphi^{[2]}_h \quad \text{so that} \quad \Phi^*_h = \varphi^{[2]}_h \circ \varphi^{[1]}_h. \tag{1.4.9}$$

A necessary and sufficient condition for the existence of α_i and β_i satisfying the diagram given in Figure 1.4 is that $\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} b_i$, which is the consistency condition for a numerical method.

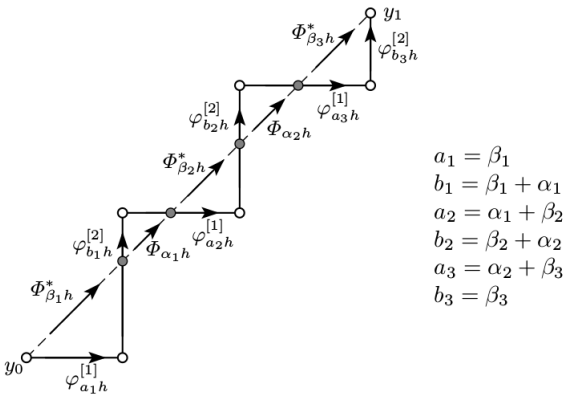


Figure 1.4: Equivalence of splitting and composition methods (Hairer et al. (2006)).

1.4.5.4 Combining exact and numerical flows

If the splitting is such that only the flow of $y' = f^{[1]}(y)$ can be computed exactly, we can consider

$$\Phi_h = \varphi^{[1]}_h \circ \Phi^{[2]}_h, \qquad \Phi^*_h = \Phi^{[2]*}_h \circ \varphi^{[1]}_h$$

as the basis of a composition method. Here $\varphi^{[1]}_h$ is the exact flow of $y' = f^{[1]}(y)$ and $\Phi^{[2]}_h$ is some first-order integration applied to $y' = f^{[2]}(y)$. The above interpretation of splitting methods as composition methods implies that the resulting method

$$\Psi_h = \varphi^{[1]}_{\alpha_s h} \circ \Phi^{[2]}_{\alpha_s h} \circ \Phi^{[2]*}_{\alpha_s h} \circ \varphi^{[1]}_{(\beta_s + \alpha_{s-1})h} \circ \Phi^{[2]}_{\alpha_{s-1} h} \circ \cdots \circ \Phi^{[2]*}_{\beta_1 h} \circ \varphi^{[1]}_{\beta_1 h}$$

has the desired high order.

1.4.5.5 Splitting into more than two vector fields

Consider a differential equation

$$y' = f^{[1]}(y) + f^{[2]}(y) + \cdots + f^{[N]}(y), \tag{1.4.10}$$

where we assume that the flow $\varphi^{[j]}(y)$ of the individual problem $y' = f^{[j]}(y)$ can be computed exactly. We can compose the first-order methods

$$\Phi_h = \varphi_h^{[1]} \circ \varphi_h^{[2]} \circ \cdots \circ \varphi_h^{[N]}$$

together with its adjoint to get a splitting method for (1.4.10) of arbitrary high level order (Hairer et al. (2006)).

Conclusion

We introduced some basic concepts of ODEs, study some numerical methods for solving ODEs. We review some concepts of Hamiltonian systems and symplectic numerical integrators. As we will see later these concepts will be relevant to optimization problems in machine learning.

2. Optimization Problems in Machine Learning and Deep Learning

Introduction

In this chapter, we present a unifying framework for machine learning as optimization problems given some examples in different fields of machine learning including supervised learning, semi-supervised learning, unsupervised learning, reinforcement learning and deep learning. This chapter includes two sections. The first section gives an overview of machine learning and its different areas of application. The second section presents problems from machine learning formulated as optimization problems.

The main references for this chapter are Géron (2019), Sun et al. (2019), LeCun et al. (2015), Bottou et al. (2018), Curtis and Scheinberg (2017).

2.1 Overview of Machine learning

Machine learning is the mechanism through which machines learn without explicitly programming them. In order to explain how optimization problem arise in machine learning, we choose the most popular class of machine learning systems based on whether or not they are trained with human supervision. This includes supervised, unsupervised, semi-supervised, and reinforcement learning.

Figure 2.1 shows the global picture of machine learning as subset of artificial intelligence and its different fields.

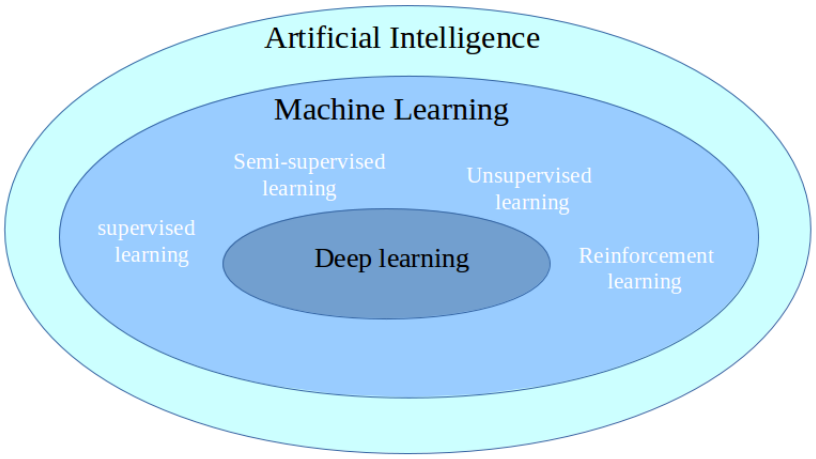


Figure 2.1: Global picture of machine learning as subset of artificial intelligence.

2.1.1 Supervised Learning

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. The main objective is to learn a mapping or association between input data samples and their corresponding outputs based on multiple training data instances. This learned knowledge can be used in the future to predict an output for any new input data sample which was previously unknown or unseen during the model training process. These methods are termed as supervised because the model learns on data samples, where the desired output responses/labels are already known beforehand in the training phase. Supervised learning problems can be further grouped into classification and regression problems. The main difference between classification and regression problems is the nature of outputs that is categorical for classification problems and numeric for regression problems.

2.1.2 Unsupervised Learning

Unsupervised learning is a machine learning technique in which previously unknown patterns in data sets without pre-existing labels are identified. It draws inferences from the data for exploratory analysis, such as finding hidden patterns or grouping similar samples or separating distinct samples. The models train on data without any right responses/labels to find hidden patterns and useful signal in the data. Application of unsupervised learning included for instance clustering, dimensionality reduction, anomaly detection, association rule-Mining.

Clustering

Clustering methods find patterns of similarity and relationships among data samples and cluster them into various groups, such that each group or cluster of data samples have some similarity, based on their inherent attributes or features. These methods are completely unsupervised, because they cluster data by looking at their data features without any prior training, supervision, or knowledge about data attributes, associations, and relationships.

Dimensionality Reduction

Once we start extracting attributes or features from raw data samples, sometimes our feature space gets bloated up with a humongous number of features. This poses the challenge of analyzing and visualizing data with thousands or millions of features, making the feature space extremely complex. This is a problem with regard to training models, memory, and space constraints. Dimensionality reduction algorithms reduce the number of features or attributes for each data sample by extracting or selecting a set of principal or representative features. The most popular algorithms for dimensionality reduction are Principal Component Analysis (PCA), nearest neighbors, and discriminant analysis, etc.

Anomaly Detection

An anomaly detection or outliers detection model finds out occurrences of rare events or observations that typically do not occur normally based on historical data samples. Sometimes anomalies occur infrequently and are thus rare events. In other instances, they might not be rare but might occur in very short bursts over time, thus have specific patterns. The model is trained on the training dataset having normal, non-normal data samples. Once it learns the necessary data representations, patterns, and relations among attributes in normal samples, it would be able to identify the new data sample as anomalous or a normal data by using its learned knowledge.

Association Rule-Mining

Association rule-mining is a data mining method used to examine and analyze large transactional datasets to find patterns and rules of interest. These patterns represent interesting relationships and associations, among various items across transactions.

2.1.3 Semi-supervised Learning

As the name suggests, semi-supervised learning lies somewhere in between both supervised and unsupervised learning. It is mainly relevant in scenarios where datasets, contain both labeled and unlabeled data.

2.1.4 Reinforcement Learning

Reinforcement learning is a wing of machine learning, and a technique to learn any kind of activity that follows a sequence of actions. A reinforcement learning agent gathers the information from the environment and observes states; it then performs an action that results in a new state and a reward (that is, quantifiable feedback from the environment). This phenomenon continues until the agent is able to improve the performance beyond a certain threshold, that is, maximizing the cumulative discounted return. At each step, these actions can be chosen randomly, can be fixed, or can be supervised using a neural network. The supervision of predicting action using a deep neural network opens a new domain, called deep reinforcement learning. This forms the base of AlphaGo, Libratus, and many other breakthrough research in the field of artificial intelligence (Li (2017)).

2.1.5 Underfitting, Overfitting and Regularization

Underfitting is the inability of a model to correctly predict the labels of the data it was trained on. It means that the model makes many mistakes on the training data. One important reason for under-fitting is, the model makes simplistic assumptions about the data or the features engineered are not informative enough.

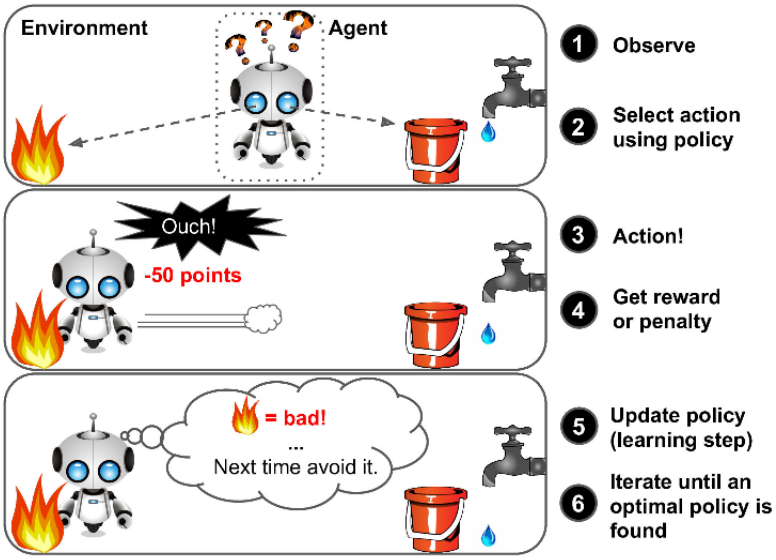


Figure 2.2: Reinforcement learning (Géron (2019)).

On the other hand, overfitting (also know as the problem of high variance) is exhibited when the model predicts very well the training data but poorly the data from at least one of the two hold-out sets. One reason for over-fitting is that the model is too complex for the data (for example a very tall decision tree or a very deep or wide neural network often over-fit); there are too many features but a small number of training examples.

A larger number of non-zeros parameters causes over-fitting. Regularization refers to a process of introducing additional information in order to prevent over-fitting. This approach constrains the model to use only a fewer number of non-zeros parameters. This information is usually of the form of a penalty for complexity of the model. A theoretical justification for regularization is that it attempts to impose *Occam's razor* on the model. The most useful regularizers are L_p-norm. In practice, p is set to 1 or 2.

2.2 Machine Learning Formulated as Optimization Problems.

In this section we explain how optimization problems arise in different aspect of machine of machine.

2.2.1 Optimization Problems in Supervised Learning

Given a set of training sample $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$, where $x^{(i)}$ is a feature vector of the ith sample and $y^{(i)}$, the corresponding label, the goal of supervised learning algorithms is to find a good mapping function $h\colon \mathcal{X} \to \mathcal{Y}$ (called the hypothesis function), where

$\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y}$ are the set of feature and label respectively, to minimize the loss function $L\colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that measures how far the predicted value $h(x)$ is from the real respective value y. This problem can be formulated as

$$\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} L(h_\theta(x^{(i)}), y^{(i)}),$$

where h_θ a mapping function parametrized by θ.

In order to alleviate over-fitting, a regularization components is usually added. The preceding optimization problem can then be formulate as

$$\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} L(h_\theta(x^{(i)}), y^{(i)}) + \lambda R(\theta),$$

where $R(\theta)$ is a regularization function and λ is a regularization parameter which can be found through cross-validation. The kind of loss functions in supervised learning, include the square of Euclidean distance, cross-entropy, contrast loss, hinge loss, information gain, etc. depending on the task (regression or classification). Let us give some examples.

Example 2.2.1. (**Linear regression**) Linear regression is a well known and most useful model in supervised learning for regression problem.

In this case the predicting function is defined by $h_\theta(x^{(i)}) = \theta_0 + \sum_{k=1}^{d} \theta_k x_k^{(i)} := \sum_{k=0}^{d} \theta_k x_k^{(i)}$, where we set for all $i = 1, \ldots, n$, $x_0^{(i)} = 1$. The loss function can be the square of the Euclidean distance. Then considering the L_2-norm as regularizer, finding the linear regression parameter lead to the following optimization problem

$$\min_{\theta \in \mathbb{R}^{d+1}} \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \sum_{k=0}^{d} \theta_k x_k^{(i)} \right)^2 + \lambda \|\theta\|_2^2.$$

Example 2.2.2. (**Linear SVM**)

Support Vector Machine (SVM) is a useful model for linear classification task in machine learning. In SVM, the predicting function is defined by $h_\theta(x^{(i)}) = \theta^T x^{(i)}$, where θ^T denotes the transpose of the vector θ. The loss function is defined as $\max(0, 1 - y^{(i)} \theta^T x^{(i)})$. This lead to the following optimization problem

$$\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} \max(0, 1 - y^{(i)} \theta^T x^{(i)}) + \lambda \|\theta\|_2^2.$$

2.2.2 Optimization Problems in Unsupervised Learning

Unsupervised learning is a machine learning class that aims to discover and find out patterns in given unlabeled data. There are so many unsupervised learning algorithm in the literature that we

cannot explore all of them in this document. We will discuss clustering algorithm and dimension reduction algorithm to explain how optimization problem arise in these cases.

Data clustering is a class of unsupervised learning approach that aims to divide data into different disjoint groups (called clusters) which are homogeneous and well separated. Homogeneity indicates the similarity of the observations within the same cluster. Given the set of unlabeled examples $\left\{x^{(i)}\right\}_{i=1}^{n}$, where each $x^{(i)}$ is a d-dimensional vector, the goal of clustering is to divide this data points into K disjoint groups $C = \{C_1, \ldots, C_K\}$ minimizing the dissimilarity d_{kl} between each pair of data points $x^{(l)}$ and $x^{(k)}$. The dissimilarity d_{kl} is commonly a distance metric. For K-means clustering, the dissimilarity measure can be the square Euclidean distance. In this case, letting $\mathcal{C}$ the set of partition of $\left\{x^{(i)}\right\}_{i=1}^{n}$ into K disjoint groups, the K-means algorithm can be formulated as follow

$$\min_{C \in \mathcal{C}} \sum_{k=1}^{K} \sum_{x^{(i)} \in C_k} \|x^{(i)} - \mu_k\|_2^2 \ ,$$

where K is the number of clusters, $x^{(i)}$ is the i^{th} feature vector of samples, μ_k is the center of cluster k, and C_k is the sample set of cluster k.

Another example of clustering algorithm is K-modes clustering that is useful in clustering the categorical data. In this case, the dissimilarity between $x^{(l)}$ and $x^{(k)}$ is defined as the total mismatches of the corresponding attribute values of the objects. Formally

$$d_{kl} := d(x^{(k)}, x^{(l)}) = \sum_{j=1}^{d} \delta(x_j^{(k)}, x_j^{(l)}),$$

where

$$\delta(x_j^{(k)}, x_j^{(l)}) = \begin{cases} 1 & if \quad x_j^{(k)} = x_j^{(l)} \\ 0 & if \quad x_j^{(k)} \neq x_j^{(l)} \end{cases} \quad .$$

Let $S = \left\{x^{(i)}\right\}_{i=1}^{n}$ be a set of categorical objects described by d categorical attributes. A mode of S is a vector $Q = (q_1, \ldots, q_d)$ that minimizes

$$D(S, Q) := \sum_{i=1}^{n} d(x^{(i)}, Q),$$

here Q is not necessarily an object of S.

Letting $\mathcal{C}$ the set of partition of $\left\{x^{(i)}\right\}_{i=1}^{n}$ into K disjoint groups, The optimization problem for partitioning this set of n objects into K clusters $C = \{C_1, \ldots, C_k\}$ becomes

$$\min_{C \in \mathcal{C}} \sum_{k=1}^{K} \sum_{x^{(i)} \in C_k} d(x^{(i)}, Q_k),$$

where Q_k is the mode of cluster C_k.

Dimensionality reduction algorithm is a well known class of unsupervised learning. It compresses data projecting it into low-dimensional space keeping as much as possible the original information

from the initial data. The most useful dimensional reduction algorithm is principal component analysis (PCA). Given the set of unlabeled examples $\left\{x^{(i)}\right\}_{i=1}^{n}$, where each feature $x^{(i)}$ is d-dimensional vector. PCA projects data into d'-dimensional space (principal subspace), with $d \gg d'$, minimizing the reconstruction error. In other words, PCA aims to reconstruct each data point $x^{(i)}$ as $\bar{x}^{(i)} := U_{d'}U_{d'}^{T}x^{(i)}$, where $U_{d'} \in \mathbb{R}^{d\times d'}$ such that $U_{d'}^{T}U_{d'} = I_{d'}$, kipping as well as possible the information containing in $x^{(i)}$ (Ghodsi, 2006). Letting $\mathcal{O} = \left\{U \in \mathbb{R}^{d\times d'}, U^{T}U = I_{d}\right\}$, the above can be formulated as the following optimization problem

$$\min_{U_{d'}\in\mathcal{O}} \sum_{i=1}^{n} \|x^{(i)} - \bar{x}^{(i)}\|_{2}^{2} ,$$

where $\bar{x}^{(i)} := U_{d'}U_{d'}^{T}x^{(i)} = \sum_{j=1}^{d'} z_{j}^{(i)}e_{j}$ is a reconstruction of $x^{(i)}$, $z^{(i)} = \left\{z_{1}^{(i)}, \ldots, z_{d'}^{(i)}\right\}$ is the projection of $x^{(i)}$ in d'-dimensional coordinates, e_j is the standard orthogonal basis under d'-dimensional coordinates.

2.2.3 Optimization Problems in Semi-supervised Learning

Semi-supervised learning (SSL) is the method between supervised and unsupervised learning, which incorporates labeled data and unlabeled data during the training process. It can deal with different tasks including classification tasks, regression tasks, clustering tasks and dimensionality reduction tasks. Different kind of semi-supervised learning methods includes self-training, generative models, semi-supervised support vector machines (S^3VM), graph-based methods, multi-learning method and others (Zemmal et al. (2016)). We take S^3VM as an example to introduce an optimization problem in semi-supervised learning, considering the binary classification problem.

The training set consists of l labeled data $\left\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(l)}, y^{(l)})\right\}$, $y^{(i)} \in \{-1, 1\}$ and u unlabeled data $\left\{x^{(l+1)}, \ldots, x^{(n)}\right\}$, with $l + u = n$ and each $x^{(i)}$ is a d-dimensional feature vector.

S^3VM aims to construct a linear SVM using both the labeled data and unlabeled data. It starts by formulating standard SVM on labeled data and then incorporates the unlabeled data (Chapelle et al. (2008)). Formally, the linear S^3VM can be formulated as the following optimization problem which is solved over both the hyperplane parameters $(w, b) \in \mathbb{R}^{d} \times \mathbb{R}$ and a label vector $y_u = \left[y^{(l+1)}, \ldots, y^{(N)}\right]$

$$\min_{w,b,y_u} C\sum_{i=1}^{l} L(w^{T}x^{(i)} + b,\ y^{(i)}) + C^{*}\sum_{j=l+1}^{n} L(w^{T}x^{(i)} + b,\ y^{(i)}) + \lambda\|w\|^{2}$$

$$\text{subject to} \quad \sum_{i=l+1}^{n} \max(y^{(i)}, 0) = r,$$

where λ is regularization parameter, $L(.,.)$ is a loss function, the loss over labeled and unlabeled examples is weighted by two hyper-parameters C, C^{*} , which reflect confidence in labels and

in the cluster assumption respectively, r is a parameter estimated from the class ratio on the labeled set, or from prior knowledge about the classification problem. Finally, the constraint $\sum_{i=l+1}^{N} \max(y^{(i)}, 0) = r$ helps in avoiding unbalanced solutions by enforcing that a certain user-specified fraction r, of the unlabeled data should be assigned to the positive class.

2.2.4 Optimization Problems in Reinforcement Learning

Reinforcement learning (RL) is one of the most promising machine learning approaches, in which an agent learns by continually interacting with an unknown stochastic environment. In RL, the agent observes the state of the environment, and based on this state/observation takes an action. Formally, an RL problem can be represent by a Markov decision process (MDP), which is a tuple with four elements: $(\mathcal{S},\ \mathcal{A},\ P(.\mid s,a),\ r(s,a))$. Here $\mathcal{S}$ is the set of states of the environment, $\mathcal{A}$ describes the set of actions or transitions between states, $P(.\mid s,a)$ is the probability distribution of next states given the current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ and $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, in the other words, for $s \in \mathcal{A}$ and $s \in \mathcal{S}$, $r(s,a)$ is the reward of taking action a in state s. Given an environment $(\mathcal{S},\ \mathcal{A},\ P(.\mid s,a),\ r(s,a))$, the ultimate goal is to compute a strategic function between the environment states and actions that maximizes expected reward. The above can be formulated as the following optimization problem

$$\max_{\pi \in \Theta} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r(s_k, \pi(s_k))\right],$$

where $\gamma \in [0,1)$ is a discount factor, Θ is the set of all admissible deterministic policies, and $(s_0, a_0, s_1, a_1, \dots)$ is a state-action trajectory generated by the Markov chain under policy π.

2.2.5 Optimization Problems in Deep Learning

Deep learning is a subfield of Machine learning that attempts to represent high-level abstraction of data by using a Deep Neural Networks (DNNs) with multiple layers involving sequential, linear and nonlinear transformations to build computational models (LeCun et al. (2015)). These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection, etc. Nowadays there exist a variety of neural networks types according to their structure and the problem to be solved. This includes Fully-connected Neural Networks (FNNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) for a few examples. The most popular neural networks are Convolutional Neural Networks and Recurrent neural networks (Pouyanfar et al. (2018)).

Convolutional Neural Networks (CNNs)

CNNs are a type of deep learning models with each module consisting of a convolutional layer and a pooling layer (Curtis and Scheinberg (2017)). CNNs are feed-forward neural networks with

convolution calculation. These modules are often stacked up with one on top of another, or with a DNN on top of it, to form a deep model. As a class of deep learning models for learning features, CNNs learns a hierarchy of increasingly complex features. Without building handcrafted features, these methods utilize layers with convolving filters that are applied on top of pre-trained word embedding. In real world, CNNs are useful for face recognition, image classification, handwriting analysis, medical image analysis, etc. Figure 2.3 shows the basic architecture of a CNN.

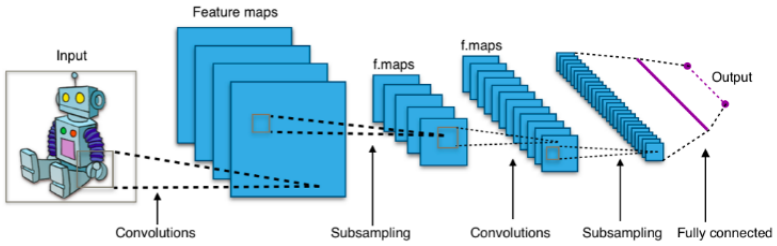


Figure 2.3: CNN architecture (Tatan).

Recurrent neural networks (RNNs)

RNNs are a type of Neural Network where the outputs from previous step are fed as inputs to the current step. It has a "memory" which remembers all information about what has been calculated (Curtis and Scheinberg (2017)). RNNs are useful in cases of sequential data, whether as an input, output, or both. The reason RNNs became so effective is because of their ability to aggregate the learning from past datasets and use that along with the new data to enhance learning. This way, it captures the sequence of events, which was not possible in a feedforward neural network nor in earlier approaches of statistical time series analysis. The following Figure 2.4 shows the basic architecture of an RNN.

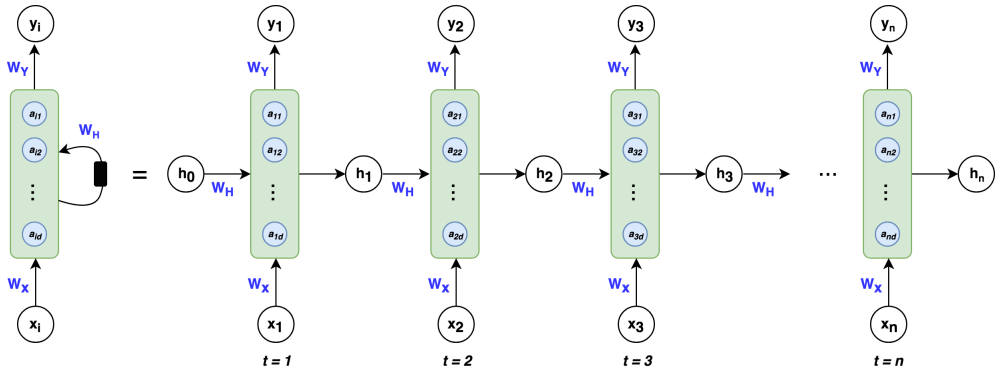


Figure 2.4: RNN architecture (Khuong) .

In this subsection we will use a DNN learning formulation in the simple case to introduce an optimization problem in deep learning. Structurally, a DNN takes the form of a graph with subsets of neurons arranged in a sequence. However, beside, the key aspect is how it performs the task. The process is performed in several steps. First, each element of an input vector

$x = (x_1, \ldots, x_n)$ is given to a different neuron in the input layer (the first layer). The values in this layer are each passed to the neurons in the next layer (first hidden layer) after multiplication with weights associated with the corresponding edges. Each hidden layer transforms the value through a linear or nonlinear activation function and passes it to the next layer after multiplication with weighs associated to the corresponding edges. The last layer (output layer) provides the predicted output.

Mathematically, assume that the DNN has M layers, with layers 1 and M being the input and output layers, respectively. Suppose that layer l, for $l = 1, 2, 3, \ldots, M$, contains d_l neurons. So d_1 is the dimension of the input data. Given an input $x \in \mathbb{R}^{d_1}$, we may then neatly summarize the action of a fully-connected neural network by letting $x_{(l)} \in \mathbb{R}^{d_l}$ denote the output, from layer l. So, we have

$$\begin{aligned} x_{(1)} &= x \in \mathbb{R}^{d_1} \\ x_{(l+1)} &= s(w^{(l)} x_{(l)} + b^{(l)}) \in \mathbb{R}^{d_{l+1}}, \quad l = 1, \ldots, M-1, \end{aligned}$$

where $w^{(l)} \in \mathbb{R}^{d_{l+1}} \times \mathbb{R}^{d_l}$ is a matrix of edge weight parameters at layer l, $b^{(l)}$ is the vector of biases for layer l, and s is a component-wise activation function. The most popular choices for the activation function include the sigmoid function $s(x) = \dfrac{1}{1+e^{-x}}$, the tangent hyperbolic function $s(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$, and the rectified linear unit function (ReLU) $s(x) = \max\{0, x\}$. In this manner, the ultimate output vector $x_{(M)}$ leads to the prediction function value $h(x; w)$, where the parameter vector w collects all the parameters $\left\{(w^{(l)}, b^{(l)})\right\}_{l=1}^{M}$ of the successive layers.

Similar to the previous section, given a set of training example $\left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{n}$, finding that parameters lead to the following optimization problem

$$\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} L(h(x^{(i)}; w), y^{(i)}),$$

where L is a loss function. According to the choice of activation function, this optimization problem can be highly nonlinear and non-convex making it more challenging to solve to global optimality.

As seen above, most machine learning algorithms lead to the following optimization problem

$$\min_{x \in \mathcal{X}} f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x) + \lambda R(x),$$

where $\mathcal{X} \subset \mathbb{R}^d$ is a convex close set and $f, f_i \colon \mathcal{X} \subset \mathbb{R}^d \to \mathbb{R}$, for $i = 1, \ldots, n$. f is typically the total loss function whereas each f_i represents the loss due to the i^{th} training sample, $R(x)$ is a regularization function and λ is a regularization parameter. Furthermore, x is a vector of trainable parameters and n is the training sample size.

Conclusion

We have illustrated how machine learning and deep learning algorithms can be formulated as a convex or non-convex optimization problem to find the extremum of an objective function. We showed how optimization problems arise in many area of machine learning including supervised learning, semi-supervised learning, unsupervised learning and deep learning. There exist many numerical methods for solving optimization problems in machine learning and deep learning. The most popular are gradient based methods including gradient descent (GD), stochastic gradient descent (SGD), Mirror descent (MD). We will consider such methods in the next chapter.

3. Ordinary Differential Equations for Gradient-based Optimization Algorithms

Introduction

In this chapter, we establish formally links between some gradient-based algorithms such as gradient descent, Polyak heavy ball, Nesterov's accelerated gradient descent, and some special ODEs. We study the accelerated methods from a continuous-time, variational point of view in which we explore the Bregman Lagrangian introduced by (Wibisono et al. (2016)), and finding the corresponding Hamiltonian ODEs. To explore the performance and the stability of symplectic integrators, we implement the Leapfrog integrator and compare it to the three-step generalized Nesterov accelerated method(a non-symplectic algorithm) to simulate the Bregman dynamic in Euclidean case.

The main references for this chapter are Betancourt et al. (2018), Jordan (2017), Lee et al. (2016), Shi et al. (2018), Shi et al. (2019), Wibisono et al. (2016).

3.1 Overview of some gradient-based methods

As seen in the previous chapter, most optimization problems in machine and deep learning can be formulated as follow

$$\min_{x\in\mathcal{X}} f(x) := \frac{1}{n}\sum_{i=1}^{n} f_i(x) + \lambda R(x), \tag{3.1.1}$$

where $\mathcal{X} \subset \mathbb{R}^d$ is a convex closed set and $f, f_i \colon \mathcal{X} \subset \mathbb{R}^d \to \mathbb{R}$, for $i = 1, \dots, n$. f is typically the total loss function whereas each f_i represents the loss due to the i^{th} training sample, $R(x)$ is a regularization function and λ is a regularization parameter. Furthermore, x is a vector of trainable parameters and n is the training sample size. We suppose f a smooth convex function ensuring the existence of single minimum. In this section we provide an overview of gradient-based methods for solving (3.1.1). We denote by $\{x_k\}_{k=0}^{\infty}$ the sequence generated by such algorithms.

3.1.1 Gradient descent

One of the most popular and simplest method for solving (3.1.1) is gradient descent. It dates back to Euler and Lagrange and is defined as follow

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

for a starting point $x_0 \in \mathbb{R}^d$, s is a step size. It achieves a $O(1/sk)$ convergence rate when ∇f is $1/s$-Lipschitz (Lee et al. (2016)). Gradient descent is very simple to implement but its convergence rate is not optimal. To improve this method, Polyak introduces the heavy-ball method (Polyak (1964)).

3.1.2 Polyak's heavy ball

The main idea of this heavy-ball method is to incorporate a momentum term into the gradient descent step as follow

$$x_{k+1} = x_k + \alpha(x_k - x_{k-1}) - s\nabla f(x_k),$$

for starting points $x_0, x_1 \in \mathbb{R}^d$, $\alpha > 0$ is the momentum coefficient and s is a step size.

While the heavy-ball method probably attains a faster rate of local convergence than gradient descent near a minimum of f, it does not come with global guarantees. Even for strongly convex function it can fail to converge for some choices of the step-size (Liu et al. (2016), Lessard et al. (2016)).

3.1.3 Nesterov's accelerated Gradient descent

Nesterov introduced a class of accelerated gradient methods that have a faster global convergence rate than gradient descent (Nesterov (1983)). The simple one, know as Nesterov's accelerated gradient descent is formulated as follow. Assuming f μ-strongly convex with L-Lipschitz gradient, Nesterov's accelerated gradient descent (NAGD-sc) in this case involves the following equations

$$y_{k+1} = x_k - s\nabla f(x_k), \quad x_{k+1} = y_{k+1} + \frac{1-\sqrt{\mu s}}{1+\sqrt{\mu s}}(y_{k+1} - y_k),$$

with starting points $x_0, y_0 \in \mathbb{R}^d$.

Equivalently, NAGD-sc can be written in a single-variable as

$$x_{k+1} = x_k + \frac{1-\sqrt{\mu s}}{1+\sqrt{\mu s}}(x_k - x_{k-1}) - s\nabla f(x_k) - \frac{1-\sqrt{\mu s}}{1+\sqrt{\mu s}}s(\nabla f(x_k) - \nabla f(x_{k-1})),$$

with $x_0 \in \mathbb{R}^n$, and $x_1 = x_0 - \dfrac{2s\nabla f(x_0)}{1+\sqrt{\mu s}}$. Nesterov proves that the NAGD-sc achieves an accelerated linear convergence rate $O((1-\sqrt{\mu/L})^k)$ if the step-size s satisfies $0 < s \leq 1/L$.

Assuming f smooth convex and not strongly convex, the Nesterov's accelerated method (NAGD-c) takes the following form

$$y_{k+1} = x_k - s\nabla f(x_k), \quad x_{k+1} = y_{k+1} + \frac{k}{k+3}(y_{k+1} - y_k),$$

for starting points $x_0, y_0 \in \mathbb{R}^n$.

This can also be written in a single variable as follow

$$x_{k+1} = x_k + \frac{k}{k+3}(x_k - x_{k-1}) - s\nabla f(x_k) - \frac{k}{k+3}s(\nabla f(x_k) - \nabla f(x_{k-1})),$$

with $x_0 \in \mathbb{R}^n$, and $x_1 = x_0 - 2s\nabla f(x_0)$. NAGD-c achieves a convergence rate $O(1/sk^2)$ for any step-size s satisfying $s \leq 1/L$. Nesterov's accelerated gradient methods achieve optimal convergence rate for smooth strongly convex functions and smooth convex functions (Nesterov (1983)).

3.1.4 Stochastic gradient descent

Solving the optimization problem (3.1.1) using the standard gradient descent requires to compute n gradient per iteration. This can be too expensive in term of time when n becomes large. An alternative to fix this problem is to use the stochastic gradient descent (SGD). The main idea is to replace the full gradient by an unbiased estimator (Bottou, 2012). In simplest form, the SGD iteration is written as

$$x_{k+1} = x_k - s\nabla f_{\eta_k}(x_k),$$

for a starting point $x_0 \in \mathbb{R}^d$ and the step size s. Here $\{\eta_k\}$ are $i.i.d$ uniform random variables taking value in $\{1, \dots, n\}$ and the step-size s is the learning rate.

Another variant of gradient descent is Mirror descent that is a generalization of classical gradient descent to non-Euclidean space using the Bregman divergence. See (Beck and Teboulle (2003), Krichene et al. (2015)) for more detail on this method and others variants of gradient-based methods (Sun et al. (2019)).

3.2 Equivalence between gradient-based methods and special ODEs.

This section is devoted to studying some gradient-based methods in a continuous-time perspective. We establish links between gradient based method such as gradient descent, Polyak heavy-ball and Nesterov's accelerated gradient descent, and special ordinary differential equations.

3.2.1 Gradient Flow ODEs

Let us start with a simple ODE. In this subsection, we derive the first-order ordinary differential equation, known as gradient flow. In addition, we show that the gradient flow is the exact limit of gradient descent method when the step-size goes to zero.

Assume that $f \in \mathcal{F}_L^1(\mathbb{R}^n)$, from the gradient descent method $x_{k+1} = x_k - s\nabla f(x_k)$ with a starting point $x_0 \in \mathbb{R}^n$, we have

$$\frac{x_{k+1} - x_k}{s} = -\nabla f(x_k). \tag{3.2.1}$$

For any non-negative integer k, letting $t_k = k/s$ and $x_k = X(t_k)$ for some smooth curve $X(t)$ defined for $t \geq 0$, we have $x_{k+1} = X(t_{k+1}) = X(t_k + s)$.

Hence, (3.2.1) can be written as $\dfrac{X(t_k + s) - X(t_k)}{s} = -\nabla f(X(t_k))$.

Taking the limit when $s \to 0$, we obtain the following ordinary differential equation known as gradient flow

$$\dot{X}(t) = -\nabla f(X(t)).$$

3.2.2 Equivalence between heavy-ball method and special ODEs

In this subsection, we will formally establish links between a Polyak's heavy-ball method and some special ordinary differential equations for a special choice of the momentum coefficient.

Recall that the heavy-ball method is defined as follows

$$x_{k+1} = x_k + \alpha(x_k - x_{k-1}) - s\nabla f(x_k), \text{ for starting points } x_0, x_1 \in \mathbb{R}^d.$$

Letting $t_k = k/\sqrt{s}$, for any non-negative integer k and $X(t)$ be a sufficiently smooth map from $[0, \infty[$ to $\mathbb{R}^n$ such that $x_k = X(t_k)$. Performing a Taylor expansion on $X(t)$ in power of $\sqrt{s}$ we obtain

$$\begin{aligned} x_{k+1} &= X(t_{k+1}) = X(t_k + \sqrt{s}) \\ &= X(t_k) + \dot{X}(t_k)\sqrt{s} + \frac{1}{2}\ddot{X}(t_k)(\sqrt{s})^2 + \frac{1}{3!}\dddot{X}(t_k)(\sqrt{s})^3 + O((\sqrt{s})^4), \end{aligned} \tag{3.2.2}$$

$$\begin{aligned} x_{k-1} &= X(t_{k-1}) = X(t_k - \sqrt{s}) \\ &= X(t_k) - \dot{X}(t_k)\sqrt{s} + \frac{1}{2}\ddot{X}(t_k)(\sqrt{s})^2 - \frac{1}{3!}\dddot{X}(t_k)(\sqrt{s})^3 + O((\sqrt{s})^4), \end{aligned} \tag{3.2.3}$$

and

$$\nabla f(X(t_k)) - \nabla f(X(t_{k-1})) = \nabla^2 f(X(t_k))\dot{X}(t_k)\sqrt{s} + O((\sqrt{s})^2). \tag{3.2.4}$$

Assuming $f \in \mathcal{S}^p_{\mu,L}(\mathbb{R}^n)$, and set $\alpha = \dfrac{1-\sqrt{\mu s}}{1+\sqrt{\mu s}}$, the heavy-ball method becomes

$$x_{k+1} = x_k + \frac{1-\sqrt{\mu s}}{1+\sqrt{\mu s}}(x_k - x_{k-1}) - s\nabla f(x_k). \tag{3.2.5}$$

Multiplying both sides of (3.2.5) by $\dfrac{1+\sqrt{\mu s}}{1-\sqrt{\mu s}}\dfrac{1}{s}$ and rearranging the equality, we obtain

$$\frac{x_{k+1} + x_{k-1} - 2x_k}{s} + \frac{2\sqrt{\mu s}}{1-\sqrt{\mu s}}\frac{x_{k+1} - x_k}{s} + \frac{1+\sqrt{\mu s}}{1-\sqrt{\mu s}}\nabla f(x_k) = 0. \tag{3.2.6}$$

Plugging the Taylor expansions (3.2.2), (3.2.3) and (3.2.4) into (3.2.6) we obtain:

$$\ddot{X}(t_k) + O(s) + \frac{2\sqrt{\mu s}}{1-\sqrt{\mu s}}\left[\dot{X}(t_k) + \frac{1}{2}\sqrt{s}\ddot{X}(t_k) + O(s)\right] + \frac{1+\sqrt{\mu s}}{1-\sqrt{\mu s}}\nabla f(X(t_k)) = 0.$$

Ignoring the $O(s)$-term we obtain the high-resolution ODE (Shi et al., 2018) for the heavy-ball method as

$$\ddot{X}(t) + 2\sqrt{\mu}\dot{X}(t) + (1+\sqrt{\mu s})\nabla f(X(t)) = 0. \tag{3.2.7}$$

Taking $s = 0$ in (3.2.7) we obtain the low-resolution ODE (Shi et al., 2018) for heavy-ball method as

$$\ddot{X}(t) + 2\sqrt{\mu}\dot{X}(t) + \nabla f(X(t)) = 0. \tag{3.2.8}$$

Thus (3.2.7) and (3.2.8) provides link between the heavy-ball method and particular ODEs.

3.2.3 Equivalence between Nesterov's accelerated gradient and special ODEs: Nesterov flow

For $f \in \mathcal{S}^p_{\mu,L}(\mathbb{R}^n)$, the NAGD-sc is written in a single variable as

$$x_{k+1} = x_k + \frac{1-\sqrt{\mu s}}{1+\sqrt{\mu s}}(x_k - x_{k-1}) - s\nabla f(x_k) - \frac{1-\sqrt{\mu s}}{1+\sqrt{\mu s}} s(\nabla f(x_k) - \nabla f(x_{k-1}), \quad (3.2.9)$$

with $x_0 \in \mathbb{R}^n$ and $x_1 = x_0 - \frac{2s\nabla f(x_0)}{1+\sqrt{\mu s}}$.

Multiplying both sides of (3.2.9) by $\frac{1+\sqrt{\mu s}}{1-\sqrt{\mu s}}\frac{1}{s}$, and rearranging the equality, we obtain

$$\frac{x_{k+1}+x_{k-1}-2x_k}{s} + \frac{2\sqrt{\mu s}}{1-\sqrt{\mu s}}\frac{x_{k+1}-x_k}{s} + (\nabla f(x_k) - \nabla f(x_{k-1}) + \frac{1+\sqrt{\mu s}}{1-\sqrt{\mu s}}\nabla f(x_k) = 0. \quad (3.2.10)$$

Plugging the Taylor expansions (3.2.2), (3.2.3) and (3.2.2) into (3.2.10) we obtain

$$\begin{aligned} \ddot{X}(t_k) + O(s) \quad &+ \quad \frac{2\sqrt{\mu s}}{1-\sqrt{\mu s}}\left[\dot{X}(t_k) + \frac{1}{2}\sqrt{s}\ddot{X}(t_k) + O(s)\right] \\ &+ \quad \nabla^2 f(X(t_k))\dot{X}(t_k)\sqrt{s} + \frac{1+\sqrt{\mu s}}{1-\sqrt{\mu s}}\nabla f(X(t_k)). \end{aligned}$$

Ignoring the $O(s)$-terms we obtain

$$\ddot{X}(t) + 2\sqrt{\mu}\dot{X}(t) + \sqrt{s}\nabla^2 f(X(t))\dot{X}(t)(1+\sqrt{\mu s})\nabla f(X(t)) = 0, \quad (3.2.11)$$

which is the high-resolution ODE for NAGD-sc (Shi et al., 2018).

Taking $s = 0$ in (3.2.11), we obtain the low-resolution ODE for NAGD-sc as follow

$$\ddot{X}(t) + 2\sqrt{\mu}\dot{X}(t) + \nabla f(X(t)) = 0,$$

which is the same as for the heavy-ball case.

For $f \in \mathcal{F}^1_L(\mathbb{R}^n)$ and non strongly convex, the Nesterov's accelerated gradient takes the form

$$x_{k+1} = x_k + \frac{k}{k+3}(x_k - x_{k-1}) - s\nabla f(x_k) - \frac{k}{k+3}s(\nabla f(x_k) - \nabla f(x_{k-1})) \quad (3.2.12)$$

with $x_0 \in \mathbb{R}^n$ given and $x_1 = x_0 - s\nabla f(x_0)$.

Using the same techniques as above, (3.2.12) can be written as

$$\frac{x_{k+1}+x_{k-1}-2x_k}{s} + \frac{3}{k}\frac{x_{k+1}-x_k}{s} + (\nabla f(x_k) - \nabla f(x_{k-1})) + (1+\frac{3}{k})\nabla f(x_k) = 0. \quad (3.2.13)$$

Plugging the Taylor expansion (3.2.2), (3.2.3) and (3.2.4) into (3.2.13) we obtain

$$\begin{aligned}\ddot{X}(t_k)+O(s) \quad &+ \quad \frac{3}{t_k}\left[\dot{X}(t_k)+\frac{1}{2}\sqrt{s}\ddot{X}(t_k)+O(s)\right] \\ &+ \quad \sqrt{s}\nabla^2 f(X(t_k))\dot{X}(t_k)\sqrt{s}+O(s)\left(1+\frac{3\sqrt{s}}{t_k}\right)\nabla f(X(t_k))=0.\end{aligned}$$

Ignoring the $O(s)$ terms, we get the high-resolution ODE for NAGD-c as (Shi et al., 2018)

$$\ddot{X}(t)+\frac{3}{t}\dot{X}(t_k)+\sqrt{s}\nabla^2 f(X(t_k))\dot{X}(t_k)+\left(1+\frac{3\sqrt{s}}{t}\right)\nabla f(X(t))=0.$$

Taking $s\to 0$, we obtain the low-resolution ODE for NAGD-c as

$$\ddot{X}(t)+\frac{3}{t}\dot{X}(t)+\nabla f(X(t))=0.$$

I the above we established links between the gradient-based methods and some special ODEs. Namely we saw that the continuous-time limit of gradient descent is the gradient flow, which is a first-order ODE, and the continuous-time limit of Polyak's heavy ball and Nesterov's accelerated gradient descent are second-order ODEs. However the functional called Bregman Lagrangian introduced in (Wibisono et al. (2016)) generates a large class of accelerated methods in continuous-time. The next section is devoted to studying some properties of the Bregman Lagrangian, showing how it generates a family of ODEs.

3.3 The Bregman Lagrangian

Let us recall that we considered the following optimization problem

$$\min_{x\in\mathcal{X}} f(x),$$

where $\mathcal{X}$ is a closed convex set and $f\colon \mathcal{X}\to\mathbb{R}$ is a continuously differentiable convex function. We consider the general non-Euclidean setting in which the space $\mathcal{X}$ is endowed with a distance generated by a function $h\colon \mathcal{X}\to\mathbb{R}$ that is convex and essentially smooth.The Bregman divergence associated with h (error of linear approximation of h) is defined as

$$D_h(y,x)=h(y)-h(x)-\langle\nabla h(x),y-x\rangle, \text{ for } x,y\in\mathcal{X}.$$

D_h is non-negative since h is convex. The Euclidean setting is obtained, taking $h(x)=\frac{1}{2}\|x\|^2$.

The Legendre conjugate of h is denoted by h^* and defined from $\mathcal{X}^*$ to $\mathbb{R}$ as

$$h^*(w)=\sup_{z\in\mathcal{X}}\{\langle w,z\rangle-h(z)\}, \qquad \text{(Wibisono et al.)}$$

where $\mathcal{X}^*$ designs the dual space of $\mathcal{X}$ i.e the space of all linear form over $\mathcal{X}$. ∇h and ∇h^* are inverses of each other and D_h satisfies $D_h(y,x)=D_{h^*}(\nabla h(x),\nabla h(y))$ (Wibisono et al. (2016)). The Bregman Lagrangian is defined as

$$\mathcal{L}(X,V,t)=e^{\alpha_t+\gamma_t}\left(D_h(X+e^{-\alpha_t}V,X)-e^{\beta_t}f(X)\right), \tag{3.3.1}$$

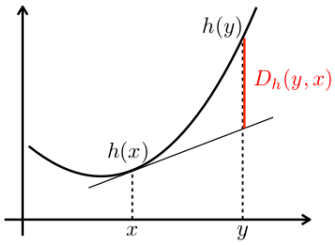


Figure 3.1: Bregman divergence associated to h (Jordan (2017)) .

where $X \in \mathcal{X}$, $V := \dot{X} \in \mathbb{R}^d$ and $t \in \mathbb{R}^+$ represent, position, velocity and time respectively (Wibisono et al. (2016)). The function $\alpha, \beta, \gamma : \mathbb{R} \to \mathbb{R}$ are arbitrary smooth increasing function of time.

α_t : weight for velocity ($e^{-\alpha}$ is "step size");

β_t : weight for potential function;

γ_t : weight for overall damping function.

(Wibisono et al. (2016)) introduced the following "ideal scaling condition", which is needed to obtain optional convergence rate

$$\begin{cases} \dot{\gamma}_t = e^{\alpha_t} \\ \dot{\beta}_t \leq e^{\alpha_t}. \end{cases} \tag{3.3.2}$$

3.3.1 Euler-Lagrange Equation for the Bregman Lagrangian

Given a general $\mathcal{L}(X_t, \dot{X}_t, t)$, one defines a functional on curves $\{X_t : t \in I\}$, where I is an interval of $\mathbb{R}$, via integration of the Lagrangian: $J = \int_I \mathcal{L}(X_t, \dot{X}_t, t)dt$. From calculus of variation, a necessary condition for a curve to minimize this functional is that it solve the following Euler-Lagrange equation

$$\frac{d}{dt}\left\{\frac{\partial \mathcal{L}}{\partial \dot{X}_t}(X_t, \dot{X}_t, t)\right\} = \frac{\partial \mathcal{L}}{\partial X_t}(X_t, \dot{X}_t, t). \tag{3.3.3}$$

Thus for the Bregman Lagrangian

$$\begin{aligned} \mathcal{L}(X_t, \dot{X}_t, t) &= e^{\alpha_t + \gamma_t}\left(D_h(X_t + e^{-\alpha_t}\dot{X}_t, X_t) - e^{\beta_t} f(X_t)\right) \\ &= e^{\alpha_t + \gamma_t}\left[h(X_t + e^{-\alpha_t}\dot{X}_t) - h(X_t) - \langle \nabla h(X_t), e^{-\alpha_t}\dot{X}_t\rangle - e^{\beta_t} f(X_t)\right]. \end{aligned}$$

We have,

$$\frac{\partial \mathcal{L}}{\partial X_t}(X_t, \dot{X}_t, t) = e^{\gamma_t + \alpha_t}\left[\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) - \nabla h(X_t) - e^{-\alpha_t}\nabla^2 h(X_t)\dot{X}_t - e^{\beta_t}\nabla f(X_t)\right].$$

$$\frac{\partial \mathcal{L}}{\partial \dot{X}_t}(X_t, \dot{X}_t, t) = e^{\gamma_t}\left[\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) - \nabla h(X_t)\right].$$

Thus,

$$\begin{aligned}\frac{d}{dt}\left\{\frac{\partial \mathcal{L}}{\partial \dot{X}_t}(X_t, \dot{X}_t, t)\right\} &= \dot{\gamma}_t e^{\gamma_t}\left[\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) - \nabla h(X_t)\right] \\ &+ e^{\gamma_t}\left[\left(\dot{X}_t - \dot{\alpha}_t e^{-\alpha_t}\dot{X}_t + e^{-\alpha_t}\ddot{X}_t\right)\nabla^2 h(X_t + e^{-\alpha_t}\dot{X}_t) - \dot{X}_t \nabla^2 h(X_t)\right].\end{aligned}$$

Plugging the above into (3.3.3), and rearranging the terms we obtain the Euler-Lagrange equation as follow

$$\begin{aligned}\ddot{X}_t + (e^{\alpha_t} - \dot{\alpha}_t)\dot{X}_t &+ e^{\alpha_t}(\dot{\gamma}_t - e^{\alpha_t})\left[\nabla^2 h(X_t + e^{-\alpha_t}\dot{X}_t)\right]^{-1}\left(\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) - \nabla h(X_t)\right) \\ &+ e^{2\alpha_t + \beta_t}\left[\nabla^2 h(X_t + e^{-\alpha_t}\dot{X}_t)\right]^{-1}\nabla f(X_t) = 0. \qquad (3.3.4)\end{aligned}$$

Now, using the ideal scaling condition (3.3.2), the second term in (3.3.4) vanishes, so the Euler-Lagrange equation simplifies to

$$\ddot{X}_t + (e^{\alpha_t} - \dot{\alpha}_t)\dot{X}_t + e^{2\alpha_t + \beta_t}\left[\nabla^2 h(X_t + e^{-\alpha_t}\dot{X}_t)\right]^{-1}\nabla f(X_t) = 0. \qquad (3.3.5)$$

In (3.3.5) we have assumed the Hessian matrix $\nabla^2 h(X_t + e^{-\alpha_t}\dot{X}_t)$ to be invertible. But we can also write it in the following way that requires only the differentiability of ∇h as

$$\frac{d}{dt}\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) = -e^{\alpha_t + \beta_t}\nabla f(X_t).$$

Theorem 3.3.1. *Under the ideal scaling condition, the solution of the Euler-Lagrange equation satisfies*

$$f(X_t) - f(x^*) \leq O(e^{-\beta_t}),$$

where $x^* = \underset{x\mathcal{X}}{argmin} f(x)$.

Proof. (see Wilson et al. (2016)) □

In particular, for the following choice of parameters

$$\alpha_t = \log p - \log t$$

$$\beta_t = p\log t + \log C \text{ , where } C > 0, \text{ and } p > 0 \text{ are constant}$$

$$\gamma_t = p\log t$$

(3.3.5) becomes $\ddot{X}_t + \frac{p+1}{t}\dot{X}_t + Cp^2 t^{p-2}\left[\nabla^2 h(X_t + \frac{t}{p}\dot{X}_t)\right]^{-1}\nabla f(X_t) = 0. \qquad (3.3.6)$

And the solution satisfies $f(X_t) - f(x^*) \leq O\left(\frac{1}{t^p}\right)$ according to the above theorem.

3.3.2 Bregman Hamiltonian and Hamiltonian system.

Recall that the Bregman Lagrangian is defined by

$$\mathcal{L}(X_t, \dot{X}_t, t) = e^{\alpha_t+\gamma_t}\left(D_h(X_t + e^{-\alpha_t}\dot{X}_t,\ X_t) - e^{\beta_t} f(X_t)\right).$$

In general, given a Lagrangian $\mathcal{L}(X_t, \dot{X}_t, t)$, the corresponding Hamiltonian is defined as

$$\mathcal{H}(X_t, \dot{X}_t, t) = \langle P_t, \dot{X}_t\rangle - \mathcal{L}(X_t, \dot{X}_t, t), \tag{3.3.7}$$

where $P_t = \dfrac{\partial \mathcal{L}}{\partial \dot{X}_t}$ is the conjugate momentum, thus Bregman Lagrangian (3.3.2),

$$P_t \ = \ \frac{\partial \mathcal{L}}{\partial \dot{X}_t} = e^{\gamma_t}\left[\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) - \nabla h(X_t)\right].$$

This implies

$$\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) = e^{-\gamma_t} P_t + \nabla h(X_t).$$

Thus

$$\nabla h^*\left(\nabla h(X_t + e^{-\alpha_t}\dot{X}_t)\right) = \nabla h^*\left(e^{-\gamma_t} P_t + \nabla h(X_t)\right).$$

and then,

$$\dot{X}_t = e^{\alpha_t}\left[\nabla h^*(\nabla h(X_t) + e^{-\gamma_t} P_t) - X_t\right]. \tag{3.3.8}$$

Now let us write the Bregman Lagrangian $\mathcal{L}(X_t, \dot{X}_t, t)$ in term of (X_t, P_t, t). We have

$$\begin{aligned}\mathcal{L}(X_t, \dot{X}_t, t) \ &= \ e^{\alpha_t+\gamma_t}\left(D_h(X_t + e^{-\alpha_t}\dot{X}_t,\ X_t) - e^{\beta_t} f(X_t)\right) \\ &= \ e^{\alpha_t+\gamma_t}\left(D_{h^*}(\nabla h(X_t), \nabla h(X_t + e^{-\gamma_t} P_t)) - e^{\beta_t} f(X_t)\right),\end{aligned} \tag{3.3.9}$$

because

$$\nabla h(X_t + e^{-\alpha_t}\dot{X}_t) = e^{-\gamma_t} P_t + \nabla h(X_t).$$

Plugging (3.3.8) into (3.3.9) and using the fact that $D_h(Y, X) = D_{h^*}(\nabla h(X), \nabla h(Y))$. and then using the definition of $D_h(.,.)$ we obtain

$$\begin{aligned}\mathcal{L}(X_t, P_t, t) \ &= \ e^{\alpha_t+\gamma_t}\left(h^*(\nabla h(X_t)) - h^*(\nabla h(X_t + e^{-\gamma_t} P_t))\right. \\ &\qquad \left. -e^{-\gamma_t}\langle \nabla h^*(\nabla h(X_t) + P_t), -e^{\gamma_t} P_t\rangle - e^{\beta_t} f(X_t)\right).\end{aligned} \tag{3.3.10}$$

Plugging (3.3.8) and (3.3.10) into (3.3.7) we obtain

$$\begin{aligned}\mathcal{H}(X_t, P_t, t) \ &= \ e^{\alpha_t+\gamma_t}\left(h^*(\nabla h(X_t + e^{-\gamma_t} P_t)) - h^*(\nabla h(X_t)) - \langle \nabla X_t, -e^{-\gamma_t} P_t\rangle + e^{\beta_t} f(X_t)\right) \\ &= \ e^{\alpha_t+\gamma_t}\left(D_{h^*}\left(\nabla h(X_t) + e^{-\gamma_t} P_t, \nabla h(X_t)\right) + e^{\beta_t} f(X_t)\right).\end{aligned} \tag{3.3.11}$$

From this Bregman Hamiltonian, we have the following corresponding Hamiltonian system know as Bregman Hamiltonian system

$$\dot{X}_t = \frac{\partial H}{\partial P_t}(X_t, P_t, t), \quad \dot{P}_t = -\frac{\partial H}{\partial X_t}(X_t, P_t, t)$$

i.e

$$\begin{aligned}
\dot{X}_t &= e^{\alpha_t}\left(\nabla h^*(\nabla h(X_t) + e^{-\gamma_t}P_t) - X_t\right) \\
\dot{P}_t &= -e^{-\alpha_t+\gamma_t}\nabla^2 h(X_t)\left(\nabla h^*(\nabla h(X_t) + e^{-\gamma_t}P_t) - X_t\right) + e^{\alpha_t}P_t - e^{\alpha_t+\beta_t+\gamma_t}\nabla f(X_t).
\end{aligned}$$

Recall that our idea in this work is to use sympletic integrators to solve optimization problems arising in machine learning. Symplectic integration of autonomous Hamiltonian systems is well known, but for non-autonomous system is less clear. In our case, the Bregman Lagrangian explicitly depends on time t, hence lifted into a time-invariant Hamiltonian that lead to a non-autonomous system. Below, we incorporate time as an explicit variable, introduce a conjugate energy variable $\mathfrak{E}$ in the configuration space to compensate the time dependence of the original Hamiltonian. Then the extended Hamiltonian, denote by H_{Ext} is defined as

$$H_{Ext}(X_t, P_t, t, \mathfrak{E}) = H(X_t, P_t, t) - \mathfrak{E},$$

where the variables X_t, P_t are conjugate as well as t, $\mathfrak{E}$. The corresponding equation of motions is written as follow

$$\begin{aligned}
\frac{dX}{d\tau} &= \frac{\partial H_{Ext}}{\partial P_t} \\
\frac{dP_t}{d\tau} &= -\frac{\partial H_{Ext}}{\partial X_t} \\
\frac{dt}{d\tau} &= \frac{\partial H_{Ext}}{\partial \mathfrak{E}} = 1 \\
\frac{d\mathfrak{E}}{d\tau} &= -\frac{\partial H_{Ext}}{\partial t}.
\end{aligned}$$

The conjugate energy $\mathfrak{E}$ compensate for the time dependence of the original Hamiltonian to ensure that the extended Hamiltonian H_{Ext} is constant along the dynamical trajectories. For more detail see (Betancourt et al. (2018), Hairer et al. (2006)).

3.4 Numerical experiments

In this section, we build a symplectic integrator, a simple Leapfrog integrator to simulate the Bregman dynamics and compare its performance with the Nesterov discretization. For this numerical experiment we consider the Euclidean case. Let $\mathcal{X}$ be a 10-dimension Euclidean space, $h(x) = \frac{1}{2}\langle x, x\rangle$. In this case, $h^* = h$, $D_h(y, x) = \frac{1}{2}\|y - x\|^2$ and both ∇h, ∇h^* are the identity function. We will use a Leapfrog integrator to find the minimum of the quadratic function $f(x) = \langle \Sigma^{-1}x, x\rangle$, where $\Sigma_{ij} = \rho^{|i-j|}, \rho = 0.9$. Our extended Hamiltonian can be simplified as

$$H_{Ext}(X_t, P_t, t, \mathfrak{E}) = \frac{1}{2}e^{\gamma_t+\alpha_t}\langle P_t, P_t\rangle + e^{\alpha_t+\beta+\gamma_t}f(X_t) - \mathfrak{E}.$$

In order to derive our symplectic integrator, we split the extended Hamiltonian into component Hamiltonians whose dynamics can be solved exactly, or at least sufficiently close to exactly numerically. Then one composes those dynamics together symmetrically (Hairer et al. (2006), Leimkuhler and Reich (2004), Betancourt et al. (2018)). For our extended Hamiltonian we can consider the splitting

$$H_{Ext} = H_A + H_B + H_C,$$

where

$$\begin{aligned} H_A(\mathfrak{E}) &= \mathfrak{E} \\ H_B(P_t, t) &= \frac{1}{2} e^{\gamma_t + \alpha_t} \langle P_t, P_t \rangle \\ H_C(X_t, t) &= e^{\alpha_t + \gamma_t + \beta_t} f(X_t). \end{aligned}$$

Using the ideal scaling conditions given in Section 3.3.1, we define the second-order Leapfrog integrator for our Bregman dynamic as follows

$$\begin{aligned} t_{n+1/2} &= t_n + \epsilon \\ \mathfrak{E}_{n+1/2} &= \mathfrak{E}_n + \epsilon \left(\frac{1}{2} \frac{p(p+1)}{t_{n+1/2}^{p+2}} \langle P_n, P_n \rangle - Cp(2p-1) t_{n+1/2}^{2p-2} f(x_n) \right) \\ P_{n+1/2} &= P_n - \epsilon C p t_{n+1/2}^{2p-1} \nabla f(x_n) \\ x_{n+1} &= x_n + \epsilon \frac{p}{t_{n+1/2}^{p+1}} P_{n+1/2} \\ P_{n+1} &= P_{n+1/2} - \epsilon C p t_{n+1/2}^{2p-1} \nabla f(x_{n+1}) \\ \mathfrak{E}_{n+1} &= \mathfrak{E}_{n+1/2} + \epsilon \left(\frac{1}{2} \frac{p(p+1)}{t_{n+1/2}^{p+2}} \langle P_{n+1}, P_{n+1} \rangle - Cp(2p-1) t_{n+1/2}^{2p-2} f(x_{n+1}) \right) \\ t_{n+1} &= t_{n+1/2} + \epsilon, \end{aligned}$$

where ϵ is the step size.

To compare the performance of the above symplectic integrators, we implement also the three-step dynamical Nesterov discretization derived in (Wibisono et al. (2016)). Considering the same scaling conditions, this algorithm is define as (Wibisono et al., 2016)

$$\begin{aligned} x_{n+1} &= \frac{p}{n+p} z_n + \frac{n}{n+p} y_n \\ y_{n+1} &= x_{n+1} - \frac{p \epsilon^p}{2N} \nabla f(x_{n+1}) \\ z_{n+1} &= z_n - \epsilon^p C p (n+1)^{p+1} \nabla f(y_{n+1}). \end{aligned}$$

The results of our simulations are presented in the figures below.

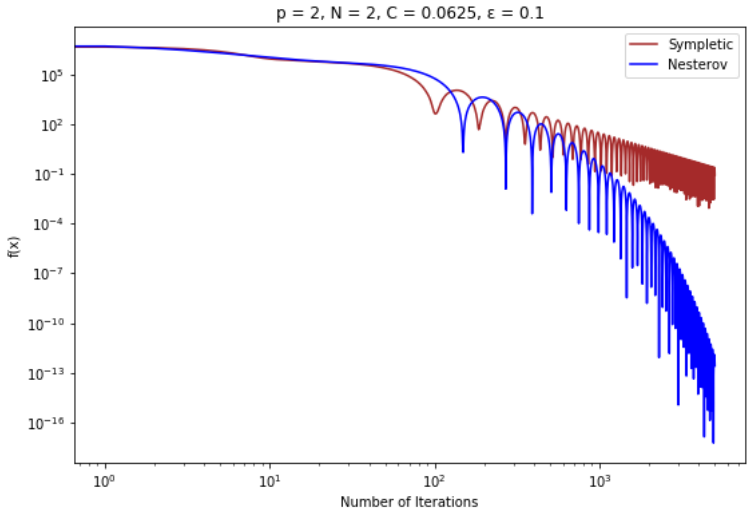


Figure 3.2: Both Leapfrog integrator and three-step generalized Nesterov discretization converge to the minimum with the similar convergence rate.

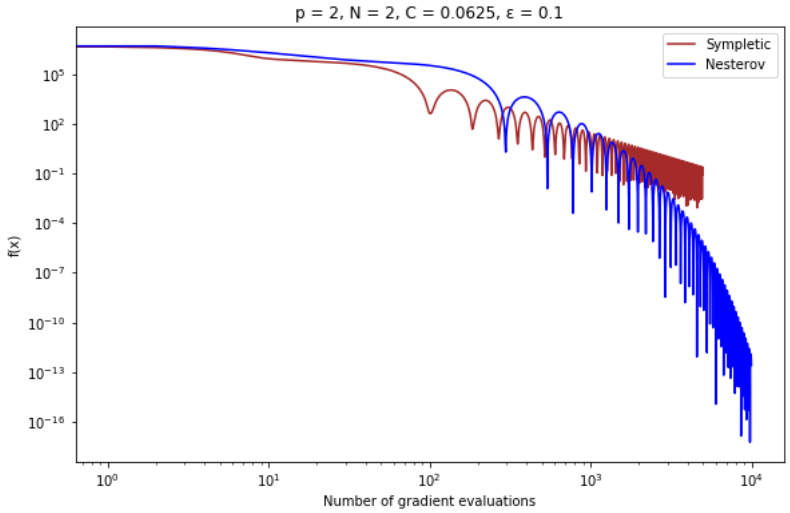


Figure 3.3: Leapfrog integrator requires only half of the computational effort of the three-step generalized Nesterov discretization.

In Figure 3.2 we can see that, both Leapfrog integrator and three-step generalized Nesterov discretization converge to the minimum with the similar convergence rate for small step-size ($\epsilon = 0.1$). However the number of iterations to arrive near the minimum is not the same for both algorithms, it is smaller for three-step generalized Nesterov discretization for this step-size. It is important to emphasize that the two algorithms don't have the same complexity. The Leapfrog integrator requires only a single gradient evaluation per iteration whereas the three-step generalized Nesterov discretization requires two to achieve stability. Hence, as shown in Figure

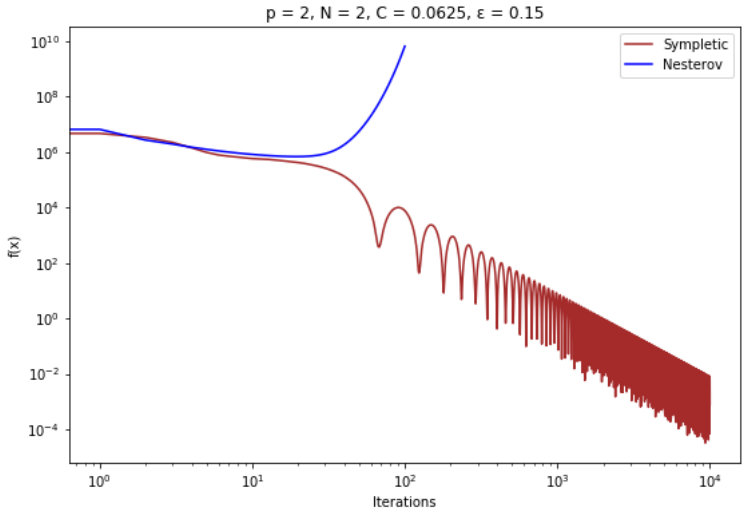


Figure 3.4: The three-step generalized Nesterov discretization quickly diverges while the Leapfrog integrator remains stable for step-size $\epsilon = 0.15$.

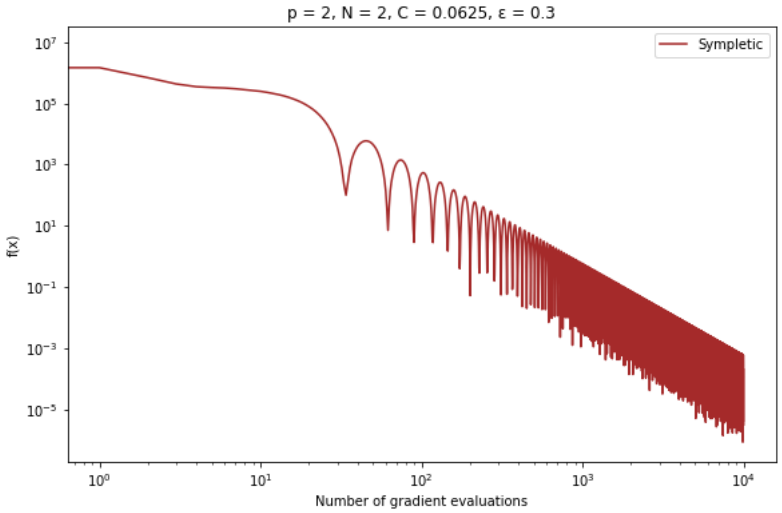


Figure 3.5: The Leapfrog integrator remains stable for step-size $\epsilon = 0.3$.

3.3, the Leapfrog integrator requires only half of the computational effort of the three-step generalized Nesterov discretization. It is also important to emphasize that due to the inherent stability of sympletic integrator (McLachlan et al. (2004), Leimkuhler and Reich (2004), Hairer et al. (2006)), we can increase the step-size with the Leapfrog integrator while for three-step generalized Nesterov discretization, there is a threshold value for the step-size to be stable. As shown in Figure 3.4 when we increase the step-size to $\epsilon = 0.15$, the three-step generalized Nesterov discretization quickly diverges while the Leapfrog integrator remains stable. Even in Figure 3.5 where we take $\epsilon = 0.3$, and the time to arrive near the minimum decreases uniformly.

Conclusion

In this chapter, we studied some gradient-based methods from the continue-time perspective. We established relationships between gradient-based methods and some special ODEs. We presented the general family of Bregman Lagrangians introduced by (Wibisono et al. (2016)), which generates a family of second-order Lagrangian dynamics that minimize the objective function at an accelerated rate. In numerical experiments, we built a symplectic integrator to simulate the Bregman dynamics and compare its performance and stability with the three-step generalized Nesterov discretization in the Euclidean case.

Conclusion and future work

Mathematical optimization is one of the most important core concept of machine learning and deep learning, arising in almost every aspect of these disciplines. The most useful numerical optimization algorithms for solving optimization problems are gradient-based methods. Such methods can be studied in continuous-time perspective via ordinary differential equations. In this thesis, we presented the basic concepts of ODEs, Hamiltonian systems and symplectic integrators. We explain how problems in machine learning and deep learning algorithms can be formulated as optimization problems. We established links between these gradient-based algorithms and special ODEs. We presented the Bregman Lagrangian, which generates a family of second-order Lagrangian dynamics that minimize the objective function at an accelerated rate. We implemented the Leapfrog integrator to simulate the Bregman dynamic and compare with tree-step generalized Nesterov's discretization in the Euclidean case.

This is only the beginning of the story and much remains to be understood. As possible future works, we plan to study the possibility to use other Hamiltonians than the Bregman Hamiltonian presented in chapter 3, study the possibility to extend the Bregman Hamiltonian in the stochastic setting, and study stochastic accelerated gradient-based algorithms in a variational framework perspective.

4. Appendix

Listing 4.1: Python code for Leapfrog and tree-step generalized Nesterov's discretization in the Euclidean case.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
ro = 0.9
d=10
def Matix_sigma(d):          # return the matrix Sigma^-1 in the expression of f(x).
        Sigma=np.zeros((d,d))
        for i in range(d):
                for j in range(d):
                Sigma[i,j]=ro**abs(i-j)
        return(np.linalg.inv(Sigma))
#def f1(x, d):                  # return f(x) for given x in R^d
#    return(sum(Matix_sigma[i, j]*x[i]*x[j] for i in range(d) for j in range(d)))

def f(x):
        return(np.inner(Matix_sigma(d).dot(x), x))

def Nabla_f(x): #  return the grdient of f at x.
        return(np.array([sum(((Matix_sigma(d)[i, j]+ Matix_sigma(d)[j, i])*x[j]) for j in
    range(d)) for i in range(d)]))

def Leapfrog(p, C, eps, iteration, t_0=0, P_0=np.zeros(d), E_0=0, x_0=np.random.randint(10,
    1000, d)):
        x=[x_0]
        f_Leap=[f(x_0)]
        P=[P_0]
        E=[E_0]
        t=[t_0]
        for n in range(1, iteration+1):
                t_half=t[n-1] + eps
                #E_half = E[n-1]+eps*(0.5*(p*(p+1)/(t_half**(p+1)))*np.inner(P[n-1], P[n-1])
     -C*p*(2*p - 1)*t_half**(2*p-2)*f(x[n-1]))
                P_half=P[n-1] - eps*C*p*t_half**(2*p-1)*Nabla_f(x[n-1])
                x_n= x[n-1]+eps*(p/t_half**(p+1)*P_half)
                x.append(x_n)
                f_Leap.append(f(x_n))
                r.append(r_half-eps*C*p*t_half**(2*p-1)*Nabla_f(x[n]))
                #E.append(E_half+eps*(0.5*(p*(p+1)/(t_half**(p+1)))*np.inner(P[n], P[n]) -C*
    p*(2*p - 1)*t_half**(2*p-2)*f(x[n])))
                t.append(t_half + eps)
                #Iterations.append(n)
        return(f_Leap)

def Tree_step_Nesterov(p, C, N, eps, iteration, y_0, z_0, x_0=np.random.randint(10, 1000, d)
    ):
        x=[x_0]
        y=[y_0]
        z=[z_0]
        f_Nest=[f(x_0)]
        N_grd_Iter=[0]
        for n in range(1, iteration+1):
                x_n = (p/n+p)*z[n-1]+(1-p/n+p)*y[n-1]
                y_n=x_n-(p*eps**p/2*N)*Nabla_f(x_n)
                z_n=z[n-1]-eps**p*C*p*n**(p-1)*Nabla_f(y_n)
                x.append(x_n)
                f_Nest.append(f(x_n))
                y.append(y_n)
                z.append(z_n)
                N_grd_Iter.append(N_grd_Iter[n-1]+2)
        return(N_grd_Iter, f_Nest)
```

References

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

Michael Betancourt, Michael I Jordan, and Ashia C Wilson. On symplectic optimization. *arXiv preprint arXiv:1802.03653*, 2018.

Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Olivier Chapelle, Vikas Sindhwani, and Sathiya S Keerthi. Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9(Feb):203–233, 2008.

David Cohen. Mini-course on geometric numerical integration. 2016.

Frank E Curtis and Katya Scheinberg. Optimization methods for supervised machine learning: From linear models to deep learning. In *Leading Developments from INFORMS Communities*, pages 89–114. INFORMS, 2017.

S Sivaji Ganesh. Lecture notes on ordinary differential equations. 2007.

Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

Ali Ghodsi. Dimensionality reduction a short tutorial. *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada*, 37:38, 2006.

Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. 2006.

Michael Jordan. On gradient-based optimization: Accelerated, distributed, asynchronous and stochastic. In *ACM SIGMETRICS Performance Evaluation Review*, volume 45, pages 58–58. ACM, 2017.

Uday Liu Kamath and John Whitaker. *Deep Learning for NLP and speech recognition*. Springer, 2019.

Ben Khuong. *The Basics of Recurrent Neural Networks (RNNs)*. URL https://medium.com/towards-artificial-intelligence/whirlwind-tour-of-rnns-a11effb7808f.

Walid Krichene, Alexandre Bayen, and Peter L Bartlett. Accelerated mirror descent in continuous and discrete time. In *Advances in neural information processing systems*, pages 2845–2853, 2015.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent only converges to minimizers. In *Conference on learning theory*, pages 1246–1257, 2016.

Benedict Leimkuhler and Sebastian Reich. *Simulating hamiltonian dynamics*, volume 14. Cambridge university press, 2004.

Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016.

Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

Jia Liu, Atilla Eryilmaz, Ness B Shroff, and Elizabeth S Bentley. Heavy-ball: A new approach to tame delay and convergence in wireless network optimization. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

Robert Mattheij and Jaap Molenaar. *Ordinary differential equations in theory and practice*, volume 43. SIAM, 1996.

Robert I McLachlan, Matthew Perlmutter, and GRW Quispel. On the nonlinear stability of symplectic integrators. *BIT Numerical Mathematics*, 44(1):99–117, 2004.

Arkadii Semenovich Nemirovsky and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pages 543–547, 1983.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and SS Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5):92, 2018.

Damien Scieur, Vincent Roulet, Francis Bach, and Alexandre d'Aspremont. Integration methods and optimization algorithms. In *Advances in Neural Information Processing Systems*, pages 1109–1118, 2017.

Bin Shi, Simon S Du, Michael I Jordan, and Weijie J Su. Understanding the acceleration phenomenon via high-resolution differential equations. *arXiv preprint arXiv:1810.08907*, 2018.

Bin Shi, Simon S Du, Weijie J Su, and Michael I Jordan. Acceleration via symplectic discretization of high-resolution differential equations. *arXiv preprint arXiv:1902.03694*, 2019.

Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems*, pages 2510–2518.

Endre Süli. Numerical solution of ordinary differential equations. *Mathematical Institute, University of Oxford*, 2010.

Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *arXiv preprint arXiv:1906.06821*, 2019.

Vincent Tatan. *Understanding CNN (Convolutional Neural Network)*. URL https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd626ee7d4.

Andre Wibisono and Ashia C Wilson. On accelerated methods in optimization. *arXiv preprint arXiv:1509.03616*, 2015.

Andre Wibisono, Ashia C Wilson, and Michael I Jordan. *Supporting Information*. URL https://www.pnas.org/content/pnas/suppl/2016/11/09/1614734113.DCSupplemental/pnas.201614734SI.pdf?targetid=nameddest%3DSTXT.

Andre Wibisono, Ashia C Wilson, and Michael I Jordan. A variational perspective on accelerated methods in optimization. *proceedings of the National Academy of Sciences*, 113(47):E7351–E7358, 2016.

Ashia C Wilson, Benjamin Recht, and Michael I Jordan. A lyapunov analysis of momentum methods in optimization. *arXiv preprint arXiv:1611.02635*, 2016.

Nawel Zemmal, Nabiha Azizi, Nilanjan Dey, and Mokhtar Sellami. Adaptative s3vm semi supervised learning with features cooperation for breast cancer classification. *Journal of Medical Imaging and Health Informatics*, 6(4):957–967, 2016.