Practical Perspectives on Symplectic Accelerated Optimization

Valentin Duruisseaux and Melvin Leok

Department of Mathematics, University of California San Diego, La Jolla, CA 92093-0112, USA.

ARTICLE HISTORY
Compiled April 11, 2023

ABSTRACT
Geometric numerical integration has recently been exploited to design symplectic accelerated optimization algorithms by simulating the Bregman Lagrangian and Hamiltonian systems from the variational framework introduced in Wibisono et al.. In this paper, we discuss practical considerations which can significantly boost the computational performance of these optimization algorithms, and considerably simplify the tuning process. In particular, we investigate how momentum restarting schemes ameliorate computational efficiency and robustness by reducing the undesirable effect of oscillations, and ease the tuning process by making time-adaptivity superfluous. We also discuss how temporal looping helps avoiding instability issues caused by numerical precision, without harming the computational efficiency of the algorithms. Finally, we compare the efficiency and robustness of different geometric integration techniques, and study the effects of the different parameters in the algorithms to inform and simplify tuning in practice. From this paper emerge symplectic accelerated optimization algorithms whose computational efficiency, stability and robustness have been improved, and which are now much simpler to use and tune for practical applications.

KEYWORDS
Accelerated optimization, Lagrangian dynamics, Hamiltonian dynamics, Bregman dynamics, symplectic integrators, variational integrators, geometric numerical integration.

1. Introduction

The field of symplectic optimization grew out of efforts to generalize Nesterov's accelerated gradient method [63], which was shown to converge in $\mathcal{O}(1/k^2)$ to the minimum of the convex objective function f and improves on the $\mathcal{O}(1/k)$ convergence rate exhibited by standard gradient descent methods. This $\mathcal{O}(1/k^2)$ convergence rate, referred to as acceleration, was shown in [64] to be optimal among first-order methods using only information about ∇f at consecutive iterates. Nesterov's algorithm was shown in [77] to limit to a second-order ordinary differential equation (ODE), as the timestep goes to 0, and that $f(x(t))$ converges to its optimal value at a rate of $\mathcal{O}(1/t^2)$ along any trajectory $x(t)$ of this ODE. It was then shown in [84] that in continuous time, an arbitrary convergence rate $\mathcal{O}(1/t^p)$ can be achieved in normed spaces, by considering flow maps generated by a family of time-dependent Bregman Lagrangian and Hamiltonian systems which is closed under time-rescaling. This lead to the field of symplectic

CONTACT Melvin Leok. Email: mleok@ucsd.edu

optimization [47], where symplectic discretizations of the Bregman Hamiltonian flow are used to construct accelerated optimization algorithms.

Lagrangian and Hamiltonian flows can also be described variationally. This, together with the time-rescaling property of this family, were exploited in [32] by using time-adaptive geometric integrators to design efficient explicit algorithms for symplectic accelerated optimization. It was observed that a careful use of adaptivity and symplecticity could result in a significant gain in computational efficiency. There has also been work on deriving accelerated optimization algorithms in the Riemannian manifold setting [1, 3–5, 28–31, 59, 88, 89].

While the symplectic optimization approach provides a broad framework for constructing accelerated optimization algorithms, the real-world performance of these methods depend on the choice of numerous parameters. In this paper, we will perform a systematic and comprehensive test of a class of symplectic accelerated optimization algorithms, so as to provide practical guidance on how to achieve good real-world performance with less tuning.

Outline of the paper.

After reviewing the basics of geometric integration in Section 2, we introduce variational accelerated optimization and present how to integrate the corresponding dynamics in Sections 3 and 4. We then analyze the oscillatory behavior of these dynamical systems in Section 5, and discuss how their unfavorable effect can be neutralized, in particular via the use of momentum restarting techniques which can dramatically improve computational efficiency and robustness. We will see that momentum restarting makes time-adaptivity futile, which allows us to simplify the algorithms. We will then compare different geometric integrators, and investigate how the computational performance depends on the different parameters, which will allow us to reduce the numbers of parameters to tune in practice. In Section 7, we see that temporal looping can avoid instability issues due to numerical precision, and finally in Section 8, we test the resulting algorithms on problems of interest to the machine learning community. For brevity and conciseness of exposition, we will move the more experimental parts of the discussion which are less theoretically interesting to Supplementary Materials.

2. Geometric Mechanics and Geometric Numerical Integration

2.1. Lagrangian and Hamiltonian Mechanics

Given a manifold $\mathcal{Q}$, a **Lagrangian** is a function $L : T\mathcal{Q} \to \mathbb{R}$. The corresponding action integral $\mathcal{S}$ is the functional $\mathcal{S}(q) = \int_0^T L(q, \dot{q})dt$, over the space of smooth curves $q : [0, T] \to \mathcal{Q}$. Hamilton's variational principle states that $\delta\mathcal{S} = 0$ where the variation $\delta\mathcal{S}$ is induced by an infinitesimal variation δq of the trajectory q that vanishes at the endpoints. Given local coordinates $(q^1, \dots, q^n)$ on the manifold $\mathcal{Q}$, Hamilton's variational principle can be shown to be equivalent to the **Euler–Lagrange equations**,

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}^k}\right) = \frac{\partial L}{\partial q^k}, \qquad \text{for } k = 1, \dots, n. \tag{1}$$

A Lagrangian L is hyperregular if the Legendre transform $\mathbb{F}L : T\mathcal{Q} \to T^*\mathcal{Q}$ of L, defined fiberwise by $\mathbb{F}L : (q^i, \dot{q}^i) \mapsto \left(q^i, \frac{\partial L}{\partial \dot{q}^i}\right)$, is diffeomorphic.

A hyperregular Lagrangian on $T\mathcal{Q}$ induces a **Hamiltonian** system on $T^*\mathcal{Q}$ via

$$H(q,p) = \langle \mathbb{F}L(q,\dot{q}),\dot{q}\rangle - L(q,\dot{q}) = \sum_{j=1}^{n} p_j\dot{q}^j - L(q,\dot{q})\Big|_{p_i=\frac{\partial L}{\partial \dot{q}^i}}, \tag{2}$$

where $p_i = \frac{\partial L}{\partial \dot{q}^i} \in T^*\mathcal{Q}$ is the conjugate momentum of q^i. There is a Hamiltonian variational principle on the Hamiltonian side in momentum phase space which is equivalent to **Hamilton's equations**,

$$\dot{p}_k = -\frac{\partial H}{\partial q^k}(p,q), \qquad \dot{q}^k = \frac{\partial H}{\partial p_k}(p,q), \qquad \text{for } k = 1,\dots,n. \tag{3}$$

and these equations are equivalent to the Euler–Lagrange equations (1), provided the Lagrangian is hyperregular. Hamiltonian systems possess a long list of structural invariants and constants of motion, the most important of which are the conservation of the Hamiltonian energy and the conservation of the symplectic 2-form.

2.2. Symplectic and Variational Integrators

Symplectic integrators form a class of geometric numerical integrators of interest since, when applied to Hamiltonian systems, they yield discrete approximations of the flow that preserve the symplectic 2-form. The preservation of the symplectic 2-form results in the preservation of many qualitative aspects of the underlying dynamical system. In particular, the numerical solution of a Hamiltonian system obtained using a constant time-step symplectic integrator is exponentially near to the exact solution of a nearby Hamiltonian system for an exponentially long time [14, 43]. It explains why symplectic integrators exhibit good energy conservation with essentially no accumulation of errors in time, when applied to Hamiltonian systems, and why symplectic methods are best suited to integrate Hamiltonian systems. We refer the reader to [45] for a brief recent overview of geometric numerical integration, and to [17, 43, 53] for a more comprehensive presentation of structure-preserving integration techniques.

Variational integrators form a class of symplectic integrators, derived by discretizing Hamilton's principle instead of discretizing Hamilton's equations directly. As a result, variational integrators are symplectic, preserve many invariants and momentum maps, and have excellent long-time near-energy preservation [60]. Traditionally, variational integrators have been designed based on the Type I generating function known as the discrete Lagrangian, $L_d : Q \times Q \to \mathbb{R}$. The exact discrete Lagrangian that generates the time-h flow of Hamilton's equations can be represented both in a variational form and boundary-value form. The latter is given by $L_d^E(q_0,q_1;h) = \int_0^h L(q(t),\dot{q}(t))dt$, where $q(0) = q_0$, $q(h) = q_1$, and q satisfies the Euler–Lagrange equations over the time interval $[0,h]$. A variational integrator is defined by constructing an approximation $L_d : Q \times Q \to \mathbb{R}$ to L_d^E, and then applying the discrete Euler–Lagrange equations,

$$p_k = -D_1 L_d(q_k,q_{k+1}), \qquad p_{k+1} = D_2 L_d(q_k,q_{k+1}), \tag{4}$$

where D_i denotes a partial derivative with respect to the i-th argument. The error analysis is greatly simplified via Theorem 2.3.1 of [60], which states that if a discrete Lagrangian, $L_d : Q\times Q \to \mathbb{R}$, approximates the exact discrete Lagrangian $L_d^E : Q\times Q \to \mathbb{R}$ to order r, i.e., $L_d(q_0,q_1;h) = L_d^E(q_0,q_1;h) + \mathcal{O}(h^{r+1})$, then the discrete Hamiltonian

map $\tilde{F}_{L_d} : (q_k, p_k) \mapsto (q_{k+1}, p_{k+1})$ defined by (4) and viewed as a one-step method, has order of accuracy r. Many properties of the integrator can be determined by analyzing the associated discrete Lagrangian, as opposed to analyzing the integrator directly.

Variational integrators have been extended to Type II/III generating functions, referred to as discrete Hamiltonians [50, 56, 74]. Hamiltonian variational integrators are derived by discretizing Hamilton's phase space principle. The boundary-value formulation of the Type II generating function of the Hamiltonian flow is given by the exact discrete right Hamiltonian, $H_d^{+,E}(q_0, p_1; h) = p_1^\top q_1 - \int_0^h \left[p(t)^\top \dot{q}(t) - H(q(t), p(t))\right] dt$, where (q, p) satisfies Hamilton's equations with boundary conditions $q(0) = q_0$, $p(h) = p_1$. A Type II Hamiltonian variational integrator is constructed by using an approximate discrete Hamiltonian H_d^+, and applying the discrete right Hamilton's equations

$$p_0 = D_1 H_d^+(q_0, p_1), \qquad q_1 = D_2 H_d^+(q_0, p_1). \tag{5}$$

Theorem 2.3.1 of [60], which simplifies the error analysis for Lagrangian variational integrators, has an analogue for Hamiltonian variational integrators. Theorem 2.2 in [74] states that if a discrete right Hamiltonian H_d^+ approximates the exact discrete right Hamiltonian $H_d^{+,E}$ to order r, i.e., $H_d^+(q_0, p_1; h) = H_d^{+,E}(q_0, p_1; h) + \mathcal{O}(h^{r+1})$, then the discrete right Hamiltonian map $\tilde{F}_{H_d^+} : (q_k, p_k) \mapsto (q_{k+1}, p_{k+1})$ defined by (5) and viewed as a one-step method, is order r accurate. Note that discrete left Hamiltonians and corresponding discrete left Hamilton's maps can also be constructed in the Type III case (see [32, 56]). Examples of variational integrators include Galerkin variational integrators [56, 60], Taylor variational integrators [75], and prolongation-collocation variational integrators [54]. In this paper, we will use Taylor variational integrators, where a discrete approximate Lagrangian or Hamiltonian is constructed by approximating the flow map and the trajectory associated with the boundary values using a Taylor method, and approximating the integral by a quadrature rule. The Taylor variational integrator is generated by the implicit discrete Euler–Lagrange equations associated to the discrete Lagrangian or by the Hamilton's equations associated with the discrete Hamiltonian. The construction of Taylor variational integrator is presented in the context of accelerated optimization in [29, 32].

In many cases, the Type I and Type II/III approaches produce equivalent integrators, such as for Taylor variational integrators provided the Lagrangian is hyperregular [75]. However, Hamiltonian and Lagrangian variational integrators are not always equivalent in practice, even when they are analytically equivalent, as they might still have different numerical properties because of numerical conditioning issues [74]. Even more to the point, Lagrangian variational integrators cannot always be constructed when the underlying Hamiltonian is degenerate, which is the case in the adaptive Hamiltonian framework for accelerated optimization presented in Section 3.4.2.

3. Variational Framework for Accelerated Optimization

3.1. General Framework

Efficient optimization has become one of the major concerns in data analysis. Many machine learning algorithms are designed around the minimization of a loss function or the maximization of a likelihood function. Due to the ever-growing size of data sets and problems, there has been a lot of focus on first-order optimization algorithms because of their low cost per iteration, and many gradient-based optimization methods have

been proposed since Cauchy's first gradient descent algorithm [22]. In 1983, Nesterov introduced an algorithm, Nesterov's Accelerated Gradient (NAG) method, which converges in $\mathcal{O}(1/k^2)$ to the minimum of the convex objective function f, improving on the $\mathcal{O}(1/k)$ convergence rate exhibited by the standard gradient descent methods. This $\mathcal{O}(1/k^2)$ convergence rate was shown in [64] to be optimal among first-order methods using only information about ∇f at consecutive iterates. This phenomenon in which an algorithm displays this improved rate of convergence is referred to as acceleration, and other accelerated algorithms have been derived, such as accelerated mirror descent [62], and accelerated cubic-regularized Newton's method [65].

It was shown in [77] that Nesterov's method limits to a second order ODE, as the step size goes to 0. The authors also proved that the objective function $f(x(t))$ converges to its optimal value at a rate of $\mathcal{O}(1/t^2)$ along the trajectories of this ODE. It was then shown in [84] that in continuous time, the convergence rate of $f(x(t))$ can be accelerated to an arbitrary rate $\mathcal{O}(1/t^p)$, by considering flow maps generated by a family of time-dependent Bregman Lagrangian and Hamiltonian systems on normed vector spaces which is closed under time rescaling. More precisely, in a general space $\mathcal{Q}$, given a convex, continuously differentiable function $h : \mathcal{Q} \to \mathbb{R}$ such that $\|\nabla h(q)\| \to \infty$ as $\|q\| \to \infty$, its corresponding Bregman divergence is $D_h(x, y) = h(y) - h(x) - \langle \nabla h(x), y - x \rangle$. The **Bregman Lagrangian and Hamiltonian** are defined as

$$L_{\alpha,\beta,\gamma}(q, v, t) = e^{\alpha_t + \gamma_t} \left[D_h(q + e^{-\alpha_t} v, q) - e^{\beta_t} f(q) \right], \tag{6}$$

$$H_{\alpha,\beta,\gamma}(q, r, t) = e^{\alpha_t + \gamma_t} \left[D_{h^*}(\nabla h(q) + e^{-\gamma_t} r, \nabla h(q)) + e^{\beta_t} f(q) \right], \tag{7}$$

which are scalar-valued functions of position $q \in \mathcal{Q}$, velocity $v \in \mathbb{R}^d$, momentum $r \in \mathbb{R}^d$, and time t, and are parametrized by smooth functions of time, α, β, γ. Here, the function $h^* : \mathcal{Q}^* \to \mathbb{R}$ denotes the Legendre transform (or convex dual function) of h, defined by $h^*(w) = \sup_{z \in \mathcal{Q}} [\langle w, z \rangle - h(z)]$. These parameter functions α, β, γ are said to satisfy the ideal scaling conditions if

$$\dot{\beta}_t \le e^{\alpha_t} \qquad \text{and} \qquad \dot{\gamma}_t = e^{\alpha_t}. \tag{8}$$

If the ideal scaling conditions are satisfied, then Theorem 1.1 in [84] asserts that

$$f(q(t)) - f(q^*) \le \mathcal{O}(e^{-\beta_t}), \tag{9}$$

along the trajectory $q(t)$ associated with the Bregman Lagrangian (6) and Bregman Hamiltonian (7), where q^* is the desired minimizer of the objective function f.

From now on, we take $h(q) = \frac{1}{2}\langle q, q \rangle$. Assuming that the functions α, β, γ satisfy the ideal scaling conditions (8), the Bregman Lagrangian and Hamiltonian become

$$L_{\alpha,\beta,\gamma}(q, v, t) = \frac{1}{2} e^{\gamma_t - \alpha_t} \langle v, v \rangle - e^{\alpha_t + \beta_t + \gamma_t} f(q), \tag{10}$$

$$H_{\alpha,\beta,\gamma}(q, r, t) = \frac{1}{2} e^{\alpha_t - \gamma_t} \langle r, r \rangle + e^{\alpha_t + \beta_t + \gamma_t} f(q), \tag{11}$$

with corresponding Euler–Lagrange equation given by

$$\ddot{q}(t) + (e^{\alpha_t} - \dot{\alpha}_t)\, \dot{q}(t) + e^{2\alpha_t + \beta_t} \nabla f(q(t)) = 0. \tag{12}$$

3.2. Polynomial Subfamily

A subfamily of Bregman dynamics of interest, indexed by a parameter $p > 0$, is given by the choice of parameter functions

$$\alpha_t = \log p - \log t, \qquad \beta_t = p \log t + \log C, \qquad \gamma_t = p \log t, \tag{13}$$

where $C > 0$ is a constant. These parameter functions satisfy the ideal scaling conditions (8), and the corresponding Lagrangian and Hamiltonian are given by

$$L_p(q, v, t) = \frac{t^{p+1}}{2p}\langle v, v\rangle - Cpt^{2p-1} f(q), \tag{14}$$

$$H_p(q, r, t) = \frac{p}{2t^{p+1}}\langle r, r\rangle + Cpt^{2p-1} f(q), \tag{15}$$

with corresponding Euler–Lagrange equation given by

$$\ddot{q}(t) + \frac{p+1}{t}\dot{q}(t) + Cp^2 t^{p-2}\nabla f(q(t)) = 0. \tag{16}$$

From Theorem 1.1 in [84], the evolution $q(t)$ resulting from this dynamical system satisfies the convergence rate $f(q(t)) - f(q^*) \leq \mathcal{O}(1/t^p)$.

Note that this Bregman subfamily has been exploited extensively in [16, 29, 32, 84], and that the special case where $p = 2$ and $C = 1/4$ corresponds to the limiting continuous differential equation introduced in [77] for Nesterov's Accelerated Gradient method.

3.3. Exponential Subfamily

Another subfamily of Bregman dynamics of interest, indexed by a parameter $\eta > 0$, is given by the choice of parameter functions

$$\alpha_t = \log \eta, \qquad \beta_t = \eta t + \log C, \qquad \gamma_t = \eta t, \tag{17}$$

where $C > 0$ is a constant. These parameter functions satisfy the ideal scaling conditions (8), and the corresponding Lagrangian and Hamiltonian are given by

$$L^\eta(q, v, t) = \frac{e^{\eta t}}{2\eta}\langle v, v\rangle - C\eta e^{2\eta t} f(q), \tag{18}$$

$$H^\eta(q, r, t) = \frac{\eta}{2e^{\eta t}}\langle r, r\rangle + C\eta e^{2\eta t} f(q), \tag{19}$$

with corresponding Euler–Lagrange equation given by

$$\ddot{q}(t) + \eta\dot{q} + C\eta^2 e^{\eta t}\nabla f(q(t)) = 0 \tag{20}$$

From Theorem 1.1 in [84], the evolution $q(t)$ resulting from this dynamical system satisfies the convergence rate $f(q(t)) - f(q^*) \leq \mathcal{O}\left(e^{-\eta t}\right)$.

3.4. Geometric Numerical Integration of Time-rescaled Bregman dynamics

3.4.1. Time-rescaling Property of the Bregman Family

A very important property of the family of Bregman dynamics is its closure under time dilation:

Theorem 3.1 ([84]). *If $q(t)$ satisfies the Euler–Lagrange equations corresponding to the Bregman Lagrangian $L_{\alpha,\beta,\gamma}$, then the reparametrized curve $y(t) = q(\tau(t))$ satisfies the Euler–Lagrange equations corresponding to the Bregman Lagrangian*

$$L_{\tilde{\alpha},\tilde{\beta},\tilde{\gamma}}(q, v, t) = \dot{\tau}(t) L_{\alpha,\beta,\gamma}\left(q, \frac{v}{\dot{\tau}(t)}, \tau(t)\right), \tag{21}$$

where

$$\tilde{\alpha}_t = \alpha_{\tau(t)} + \log \dot{\tau}(t), \qquad \tilde{\beta}_t = \beta_{\tau(t)}, \qquad \tilde{\gamma}_t = \gamma_{\tau(t)}. \tag{22}$$

Thus, the entire subfamily of Bregman trajectories can be obtained by speeding up or slowing down along any specific Bregman curve in spacetime. It is natural to exploit this time-rescaling property with carefully chosen variable time-steps in the integrator to transform the time-dependent Bregman Hamiltonian or Lagrangian into a simpler autonomous system in some extended phase-space. This allows the higher-order Bregman dynamics to be integrated in a more computationally efficient fashion by time-rescaling the lower-order Bregman dynamics. This was first achieved in [32] with the polynomial subfamily of Section 3.2, where time-rescaling a solution to the p-Bregman Euler–Lagrange equations via $\tau(t) = t^{\mathring{p}/p}$ yielded a solution to the $\mathring{p}$-Bregman Euler–Lagrange equations. We can similarly jump from one solution of Bregman dynamics from the exponential subfamily from Section 3.3 to another via $\tau(t) = \frac{\mathring{\eta}}{\eta} t$, or jump from exponential Bregman dynamics to polynomial Bregman dynamics via $\tau(t) = \frac{p}{\eta} \log t$, and vice-versa via $\tau(t) = e^{\eta t/p}$.

However, when symplectic integrators were first used in combination with variable time-steps, they performed poorly [20, 39]. A major advantage of using symplectic integrators on conservative Hamiltonian systems is that they exhibit excellent long-time near-energy preservation [43]. Backward error analysis [43] shows that symplectic integrators can be associated with a modified Hamiltonian in the form of a formal power series in the time-step. Using variable time-step results in a different modified Hamiltonian at every iteration, which is the source of the poor energy conservation and poor overall performance of these symplectic integrators. Fortunately, there are ways to circumvent this issue, which will allow us to exploit the time-rescaling property of the Bregman dynamics with variable time-steps integrators, to transform the time-dependent Bregman dynamics into simpler autonomous systems in an extended space.

3.4.2. Time-adaptive Hamiltonian Integrators

On the Hamiltonian side, the Poincaré transformation is a way to incorporate variable time-steps in geometric Hamiltonian integrators without losing their nice conservation properties [32, 41, 87]. Given a Hamiltonian $H(q, p, t)$, consider a desired transformation of time $t \mapsto \tau$ described by the monitor function $\frac{dt}{d\tau} = g(t)$. The time t shall be referred to as the physical time of the system, while τ will be referred to as the fictive time. A

new Hamiltonian system is constructed using the Poincaré transformation,

$$\bar{H}(\bar{q},\bar{p}) = g(\mathfrak{q})\,(H(q,\mathfrak{q},p)+\mathfrak{p})\,, \tag{23}$$

in the extended phase space defined by $\bar{q} = \left[\begin{smallmatrix} q \\ \mathfrak{q} \end{smallmatrix}\right] \in \bar{\mathcal{Q}}$ and $\bar{p} = \left[\begin{smallmatrix} p \\ \mathfrak{p} \end{smallmatrix}\right]$, where $\mathfrak{p}$ is the conjugate momentum for $\mathfrak{q} = t$ with $\mathfrak{p}(0) = -H(q(0),0,p(0))$. Then, using a symplectic integrator with constant time-step in fictive time τ on the Poincaré transformed Hamiltonian, has the effect of integrating the original system with the desired variable time-step in physical time t via the relation $\frac{dt}{d\tau} = g(t)$. Note that this framework can be extended to monitor functions which also depend on position q and momentum p, $g = g(q,t,p)$ (see [32]), but we will only need $g = g(t)$ in this paper. Also note that the Poincaré transformed Hamiltonian can be thought of as coming from a variational principle [29].

Going back to accelerated optimization, and denoting momentum by r to avoid confusion, we can jump from one form of Bregman dynamics to another using one of the following monitor functions

$$g(t) = \frac{p}{\mathring{p}}t^{1-\mathring{p}/p}, \qquad g(t) = \frac{\eta}{\mathring{\eta}}, \qquad g(t) = \frac{\eta}{p}t, \qquad g(t) = \frac{p}{\eta}e^{-\eta t/p}. \tag{24}$$

3.4.3. Time-adaptive Lagrangian Integrators

The time-adaptive framework for symplectic integration on the Hamiltonian side presented in the previous section relies on a degenerate Hamiltonian which has no associated Lagrangian description. Therefore, we cannot exploit the usual correspondence between Hamiltonian and Lagrangian dynamics, and we follow a different strategy to allow time-adaptivity in Lagrangian integrators. Given a time-dependent Lagrangian $L(q,\dot{q},t)$, consider the extended autonomous Lagrangian

$$\bar{L}(\bar{q}(\tau),\bar{q}'(\tau)) = \mathfrak{q}'(\tau)L\left(q(\tau),\frac{q'(\tau)}{g(\mathfrak{q}(\tau))},\mathfrak{q}(\tau)\right) - \lambda(\tau)\left[\mathfrak{q}'(\tau) - g(\mathfrak{q}(\tau))\right], \tag{25}$$

defined in the extended space $\bar{q} = (q,\mathfrak{q},\lambda)^\intercal$ where time is viewed as a position coordinate $\mathfrak{q} = t$, where λ is a Lagrange multiplier enforcing the desired time rescaling $\frac{dt}{d\tau} = g(t)$, and where apostrophes denote derivatives with respect to τ. Now, if $(\bar{q}(\tau),\bar{q}'(\tau))$ satisfies the Euler–Lagrange equations corresponding to the Lagrangian $\bar{L}$, then its components satisfy $\frac{dt}{d\tau} = g(t)$ and the original Euler–Lagrange equations [29].

A discrete variational formulation of these continuous extended Lagrangian mechanics can be formulated [29], by considering a discrete Lagrangian

$$L_d(q_k,\mathfrak{q}_k,q_{k+1},\mathfrak{q}_{k+1}) \approx \operatorname*{ext}_{\substack{(q,\mathfrak{q})\in C^2([\tau_k,\tau_{k+1}],\mathcal{Q}\times\mathbb{R}) \\ (q,\mathfrak{q})(\tau_k)=(q_k,\mathfrak{q}_k),\ (q,\mathfrak{q})(\tau_{k+1})=(q_{k+1},\mathfrak{q}_{k+1})}} \int_{\tau_k}^{\tau_{k+1}} L\left(q,\frac{q'}{g(\mathfrak{q})},\mathfrak{q}\right)d\tau.$$

where $0 = \tau_0 < \tau_1 < \ldots < \tau_N$ partitions the time interval of interest, and $\{(q_k,\mathfrak{q}_k)\}_{k=0}^N$ is a discrete curve in $\mathcal{Q}\times\mathbb{R}$ such that $q_k \approx q(\tau_k)$ and $\mathfrak{q}_k \approx \mathfrak{q}(\tau_k)$. Defining the discrete momenta via the discrete Legendre transformations, $p_k = -D_1L_d(q_k,\mathfrak{q}_k,q_{k+1},\mathfrak{q}_{k+1})$, and using a constant time-step h in fictive time τ, the corresponding discrete extended

Euler–Lagrange equations can be written as

$$\begin{aligned} p_k &= -D_1 L_d(q_k, \mathfrak{q}_k, q_{k+1}, \mathfrak{q}_{k+1}), \\ p_{k+1} &= \frac{g(\mathfrak{q}_k)}{g(\mathfrak{q}_{k+1})} D_3 L_d(q_k, \mathfrak{q}_k, q_{k+1}, \mathfrak{q}_{k+1}), \\ \mathfrak{q}_{k+1} &= \mathfrak{q}_k + h g(\mathfrak{q}_k), \end{aligned} \tag{26}$$

with two additional equations for $\mathfrak{p}_k$ and $\mathfrak{p}_{k+1}$ (see [29]). For accelerated optimization, we are usually not interested in the evolution of $\mathfrak{p}$, and since it will not appear in the updates for the other variables we do not need these equations and will omit them here. We can then use one of the monitor functions

$$g(t) = \frac{p}{\mathring{p}} t^{1-\mathring{p}/p}, \qquad g(t) = \frac{\eta}{\mathring{\eta}}, \qquad g(t) = \frac{\eta}{p} t, \qquad g(t) = \frac{p}{\eta} e^{-\eta t/p}, \tag{27}$$

to transform from one type of Bregman Lagrangian to another.

4. Numerical Methods and Problems of Interest

4.1. Numerical Methods

We now present four different methods to design symplectic integrators for the time-rescaled Bregman Lagrangian and Bregman Hamiltonian systems presented in Section 3. Keeping in mind the desired applications in machine learning where problem sizes and data sets are very large, we restrict ourselves to explicit first-order optimization algorithms. Each of these four methods can be used within the four different adaptive approaches presented in Section 3.4.1 (polynomial, exponential, polynomial-to-exponential, and exponential-to-polynomial), and the resulting algorithms are presented in Supplementary Material 3.

*4.1.1. **H**amiltonian **T**aylor **V**ariational **I**ntegrator (**HTVI**)*

Proceeding as in [32, Section 4.4] or [75], we can derive the Hamiltonian Taylor Variational Integrator (HTVI),

$$\begin{aligned} p_{k+1} &= p_k - h D_1 H(q_k, p_{k+1}), \\ q_{k+1} &= q_k + h D_2 H(q_k, p_{k+1}). \end{aligned} \tag{28}$$

These updates recover the Symplectic Euler method [43], which is a popular symplectic integrator of order 1.

*4.1.2. **L**agrangian **T**aylor **V**ariational **I**ntegrator (**LTVI**)*

As in [29], we can define a discrete Lagrangian,

$$L_d(\bar{q}_0, \bar{q}_1) = h L_p\left(q_0, \frac{q_1 - q_0}{h g(\mathfrak{q}_0)}, \mathfrak{q}_0\right), \tag{29}$$

and the updates for the Lagrangian Taylor Variational Integrator (LTVI) can be obtained from the discrete extended Euler–Lagrange equations (26).

4.1.3. *Störmer-Verlet (SV)*

A popular symplectic integrator is the Störmer–Verlet (SV) method,

$$\begin{aligned} p_{k+1/2} &= p_k - \frac{h}{2} D_1 H(q_k, p_{k+1/2}), \\ q_{k+1} &= q_k + \frac{h}{2}\left[D_2 H(q_k, p_{k+1/2}) + D_2 H(q_{k+1}, p_{k+1/2})\right], \\ p_{k+1} &= p_{k+1/2} - \frac{h}{2} D_1 H(q_{k+1}, p_{k+1/2}), \end{aligned} \tag{30}$$

which is a symmetric symplectic integrator of order 2 (see [43]). A very detailed description of the Störmer–Verlet method, its different interpretations, and its beneficial numerical properties can be found in [42]. Note however that in the polynomial and polynomial-to-exponential frameworks, the update for $\mathfrak{q}$ in the resulting integrators becomes implicit, which makes these integrators less desirable. For the accelerated optimization application, we will usually be able to combine the first and last updates for the momentum vector p into a single update and save roughly a third of the computational time. This is because Störmer–Verlet is conjugate to symplectic Euler.

4.1.4. *Symmetric Leapfrog Composition of Component Dynamics (SLC)*

The main idea is to decompose the vector field into its components,

$$\begin{aligned} \frac{d}{d\tau} = \frac{dq}{d\tau}\frac{d}{dq} + \frac{d\mathfrak{q}}{d\tau}\frac{d}{d\mathfrak{q}} + \frac{dr}{d\tau}\frac{d}{dr} + \frac{d\mathfrak{r}}{d\tau}\frac{d}{d\mathfrak{r}} &= \frac{\partial H}{\partial r}\frac{d}{dq} + \frac{\partial H}{\partial \mathfrak{r}}\frac{d}{d\mathfrak{q}} - \frac{\partial H}{\partial q}\frac{d}{dr} - \frac{\partial H}{\partial \mathfrak{q}}\frac{d}{d\mathfrak{r}} \\ &= \mathcal{A} + \mathcal{B} + \mathcal{C} + \mathcal{D} \end{aligned}$$

and then combine the component dynamics using a symmetric leapfrog composition

$$\Phi_h = \exp\left(\frac{h}{2}\mathcal{D}\right) \circ \exp\left(\frac{h}{2}\mathcal{C}\right) \circ \exp\left(\frac{h}{2}\mathcal{B}\right) \circ \exp\left(h\mathcal{A}\right) \circ \exp\left(\frac{h}{2}\mathcal{B}\right) \circ \exp\left(\frac{h}{2}\mathcal{C}\right) \circ \exp\left(\frac{h}{2}\mathcal{D}\right)$$

which satisfies $\Phi_h = \exp\left(hH\right) + \mathcal{O}(h^3)$ (can be shown using the Baker–Campbell–Hausdorff formula). This strategy is similar to the integrator from [16, Section 3.3] and the Splitting algorithms in [32]. An explicit derivation of the SLC algorithm for the polynomial Bregman Hamiltonian is provided in Supplementary Material 2 as an example. The position of the $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ terms was chosen to save computational time, as explained with the polynomial example in Supplementary Material 2.

4.1.5. *Remarks*

It was observed in [32] that symplecticity of the integrator was essential for the efficient, robust, and stable discretization of these variational flows describing accelerated optimization. Therefore, we will not consider non-symplectic methods here.

Higher-order explicit symplectic integrators can be derived, leveraging higher-order compositions such as Yoshida splittings [86], but it was observed in [32] that these require much more evaluations of the objective function and its gradient at each step. Thus, the resulting algorithms would not be competitive in terms of computational time and number of gradient evaluations, since the other methods usually converge in a similar number of iterations but only require one gradient evaluation per iteration.

4.2. Problems of Interest

A subset C of $\mathbb{R}^d$ is **convex** if $\lambda x + (1-\lambda)y \in C$ for any $x, y \in C$ and $\lambda \in [0,1]$. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if its domain $\mathrm{dom}(f)$ is convex and for any $\lambda \in [0,1]$ and $x, y \in \mathrm{dom}(f)$, we have that $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$, or equivalently if $f(y) \geq f(x) + \nabla f(x)^\top (y - x)$ for any $x, y \in \mathrm{dom}(f)$. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is **strongly convex** if there exists $\mu > 0$ such that $f(x) - \mu\|x\|^2$ is convex, or equivalently if $f(y) \geq f(x) + \nabla f(x)^\top (y-x) + \mu\|y-x\|^2$ for any $x, y \in \mathrm{dom}(f)$.

In our numerical experiments, we will use termination criteria of the form,

$$|f(x_k) - f(x_{k-1})| < \delta \quad \text{and} \quad \|\nabla f(x_k)\| < \delta, \tag{31}$$

for various values of the tolerance δ, and solve the following convex problems.

Problem 1. Minimize the quartic polynomial

$$f(x) = 1 + \left[(x-1)^\top \Sigma (x-1)\right]^2, \quad \text{where } \Sigma_{ij} = 0.9^{|i-j|} \text{ and } x \in \mathbb{R}^d. \tag{32}$$

This convex function achieves its global minimum at $x^* = (1, 1, \ldots, 1)^\top$.

Problem 2. Minimize the convex (not strongly convex) function

$$f(x_1, x_2) = x_1 + x_2^2 - \ln(x_1 x_2), \tag{33}$$

which achieves its global minimum at $x^* = (1, \sqrt{2}/2)^\top$.

Problem 3. Minimize the strongly convex function known as negative entropy

$$f(x_1, ..., x_d) = \sum_{k=1}^{d} x_k \log x_k. \tag{34}$$

This function achieves its global minimum at $x^* = (e^{-1}, e^{-1}, \ldots, e^{-1})^\top$.

Problem 4. Minimize the ill-conditioned strongly convex function

$$f(x_1, x_2, x_3) = 1 + 0.01x_1^2 + x_2^2 + 100x_3^2, \tag{35}$$

which achieves its global minimum at $x^* = (0, 0, 0)^\top$.

Problem 5 (**Linear Regression** or **Least Squares**)**.** Given a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and a vector $b \in \mathbb{R}^m$, consider the problem of finding a vector $x \in \mathbb{R}^n$ such that $\|Ax - b\|_2$ is minimized. The least squares problem has many applications in data-fitting and interpolation. It can be formulated as the minimization of

$$f(x) = \frac{1}{2} x^\top A^\top A x - b^\top A x, \tag{36}$$

with a gradient given by $\nabla f(x) = A^\top A x - A^\top b$. A vector $x \in \mathbb{R}^n$ is a solution of the least squares problem if and only if it satisfies the normal equation $A^\top A x = A^\top b$. Furthermore, the least squares problem has a unique solution, given by $x^* = (A^\top A)^{-1} A^\top b$, if and only if the matrix A has full rank [82].

There are also regularized versions of the least squares problem or linear regression [15, 19], to penalize larger values of x. A common form of regularization is Tikhonov regularization [69, 80, 81] (or ℓ^2 regularization), where we minimize the convex function

$$f(x) = \|Ax - b\|_2^2 + \lambda \|x\|_2^2, \tag{37}$$

for some $\lambda > 0$, which has a unique minimizer $x^* = (A^\top A + \lambda b)^{-1} A^\top b$. Another regularized version is the ℓ^1 penalized linear regression (also known as the Lasso problem [79]), where we minimize the convex (not strongly convex) function

$$f(x) = \|Ax - b\|_2^2 + \lambda \|x\|_1. \tag{38}$$

Problem 6 (**Logistic Regression** for Binary Classification)**.** Given a set of feature vectors $x_1, \ldots, x_m \in \mathbb{R}^n$ and associated labels $y_1, \ldots, y_m \in \{-1, 1\}$, we want to find a vector $w \in \mathbb{R}^n$ such that $sign(w^\top x)$ is a good model for $y(x)$. This can be formulated as the problem of minimizing the convex (not strongly convex) function

$$f(w) = \sum_{i=1}^{m} \log\left(1 + \exp\left(-y_i w^\top x_i\right)\right). \tag{39}$$

As for linear regression, there are also regularized versions of logistic regression, such as ℓ^1 and ℓ^2 regularized logistic regression, obtained by adding $\lambda\|x\|_1$ or $\lambda\|x\|_2^2$.

Problem 7 (**Fermat–Weber Location Problem** [13, 18, 26])**.** Given a set of points $y_1, \ldots, y_m \in \mathbb{R}^n$ and associated positive weights $w_1, \ldots, w_m \in \mathbb{R}$, we want to find the location $x \in \mathbb{R}^n$ whose sum of weighted distances from the points $y_1, \ldots, y_m$ is minimized. In other words, we wish to minimize the convex function

$$f(x) = \sum_{j=1}^{m} w_j \|x - y_j\|. \tag{40}$$

The Fermat–Weber location problem is at the heart of Location Theory and has countless applications across many fields of science and engineering.

Remark 1. A Tikhonov-type regularization can also be achieved by modifying the second-order differential equation instead of adding a penalty to the objective function (see [2, 7, 8, 46] for instance). The idea is to add an extra term $\epsilon(t)x(t)$ with $\epsilon(t) \to 0$ as $t \to \infty$ to the second-order differential equation of interest:

$$\ddot{x}(t) + \alpha(t)\dot{x}(t) + \gamma(t)\nabla f(x(t)) + \epsilon(t)x(t) = 0. \tag{41}$$

This extra term forces the generated trajectory to converge to a solution of minimal norm. This type of modified differential equation can be generated from a variational framework via Lagrangians and Hamiltonians of the form

$$L(x, v, t) = \frac{1}{2}\alpha(t)\langle v, v\rangle + \epsilon(t)\langle v, x\rangle - \gamma(t) f(x), \tag{42}$$

$$H(x, p, t) = \frac{1}{2\alpha(t)}\langle p - \epsilon(t)x, p - \epsilon(t)x\rangle + \gamma(t) f(x), \tag{43}$$

whose Euler–Lagrange equation reads

$$\alpha(t)\ddot{x}(t) + \dot{\alpha}(t)\dot{x}(t) + \gamma(t)\nabla f(x(t)) + \dot{\epsilon}(t)x(t) = 0. \tag{44}$$

5. Controlling the Oscillatory Behavior

The Bregman Euler–Lagrange equation (12) can be written in the form

$$\ddot{x}(t) + d(t)\dot{x}(t) + b(t)\nabla f(x(t)) = 0. \tag{45}$$

The introduction of momentum in the dynamical system causes the solution to this ordinary differential equation to overshoot frequently in its path towards the minimizer of the objective function f, and as a result the solution can be highly oscillatory. Therefore, this differential equation can be thought of as modeling a nonlinear oscillator with damping, and the convergence of the function f to its minimum value is not monotone along Bregman trajectories. This is similar to what was observed for the limiting continuous differential equation for Nesterov's accelerated gradient method [61, 77] and for most momentum methods. These oscillations are problematic since they can significantly slow down optimization algorithms that are derived from the discretization of these Bregman differential equations. Indeed, to resolve the fast oscillations of the differential equation, the time-step in the discretization has to be reduced sufficiently, which can considerably increase the number of iterations and gradient evaluations needed to achieve convergence. If the time-step is not taken small enough, the momentum in the algorithms can lead to large overshoots which can result in divergence. It would therefore be desirable to have a mechanism to neutralize these oscillations. Fortunately, there are ways to reduce the effect of these oscillations, which we will discuss in the remainder of this section.

5.1. Momentum Restarting

Momentum causes the solution to the Bregman Euler–Lagrange equation to overshoot frequently on its path towards the minimizer of f. One strategy to control these overshoots and reduce the effect of the resulting oscillations is to use restarting or momentum restarting schemes, previously explored in [23, 25, 35, 36, 38, 66, 70–72, 77]. We will consider three different momentum restarting schemes:

- **Function Scheme**: Restart momentum whenever $f(q_k) > f(q_{k-1})$
  This scheme restarts the momentum r whenever the function evaluation at the new update moves away from the minimum value, to try to avoid wasting iterations in a bad direction.

- **Gradient Scheme**: Restart momentum whenever $\nabla f(q_k)(q_k - q_{k-1}) > 0$
  This restarts the momentum variable r whenever momentum seems to take the new updates in a bad direction, measured using the gradient at that point.

- **Velocity Scheme**: Restart momentum whenever $\|q_{k+1} - q_k\| < \|q_k - q_{k-1}\|$
  This scheme restarts the momentum variable r whenever the norm of the (discrete version of the) velocity $\|\dot{q}\|$ starts decreasing, to try to maintain a high velocity along the trajectory.

Note that the quantities needed to implement these restarting schemes are already calculated in the standard versions of the optimization algorithms, and thus there is a negligible difference in the computational costs of each iteration in the restarted and non-restarted schemes. We can also require a minimum number of iterations between momentum restarts, to avoid having consecutive restarts that are too close to each other. In practice however, it did not seem to really improve the computational efficiency of the algorithm and could sometimes negatively impact the overall performance. For simplicity, we will not impose a minimum number of iterations between consecutive restarts.

In our first numerical experiment, we compared the performance of the standard algorithms to their restarted versions on three different problems for fixed values of all the parameters except the time-step h. Figure 1 shows the resulting error plots after tuning the value of h optimally. We can clearly see that the restarted versions of the algorithms are much less oscillatory, and they can allow for much larger time-steps leading to significantly faster algorithms, as is the case for Problems 2 and 3. Problem 1 is a special instance where larger time-steps cannot be taken in the restarted algorithms, despite their non-oscillatory nature. It should be noted however that although the use of momentum restarting does not lead to significant improvements in computational efficiency, it does not penalize computational efficiency either.

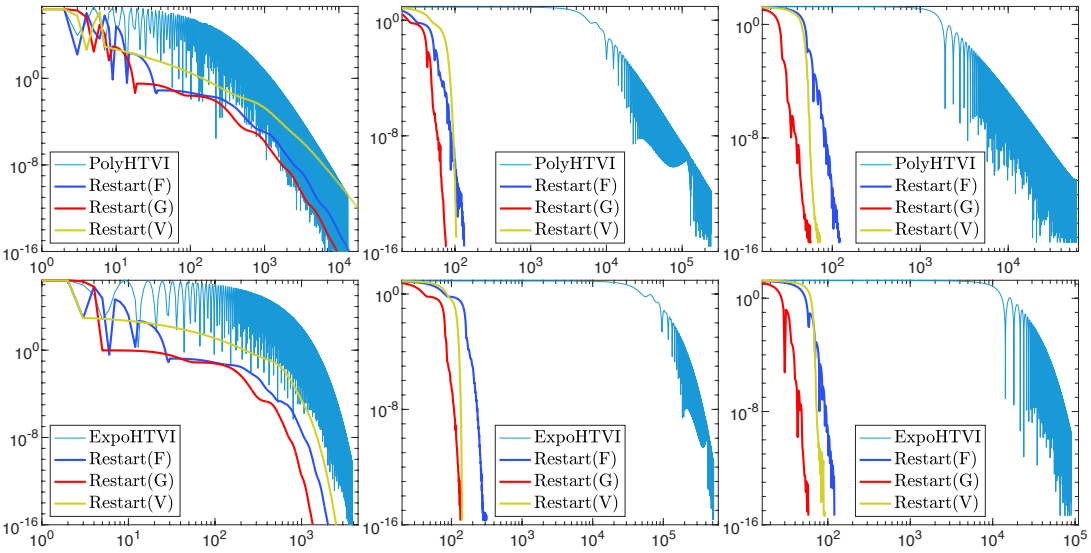

Figure 1.: Error vs. Iterations number as the standard PolyHTVI and ExpoHTVI algorithms and their restarted versions (Function (F), Gradient (G) and Velocity (V)) are applied to Problem 1 (left), Problem 2 (middle), and Problem 3 (right).

We have performed additional experiments to obtain a better idea of the benefits of momentum restarting in terms of computational efficiency, robustness and stability. More precisely, we solved optimization problems using the different versions of the algorithms on a 100×100 grid with logarithmic spacing in the parameter (C, h)-plane, and recorded the number of iterations required to achieve certain convergence criteria. Figures 2, 3, 4, and 5 display the results as filled contour plots (where the absence of color indicates either divergence or failure of the algorithm to converge in less than 10^6 iterations). Table 1 displays the number of iterations required to converge by each version of the algorithms with its optimal (C, h) pair on the 100×100 logarithmically-spaced grid.

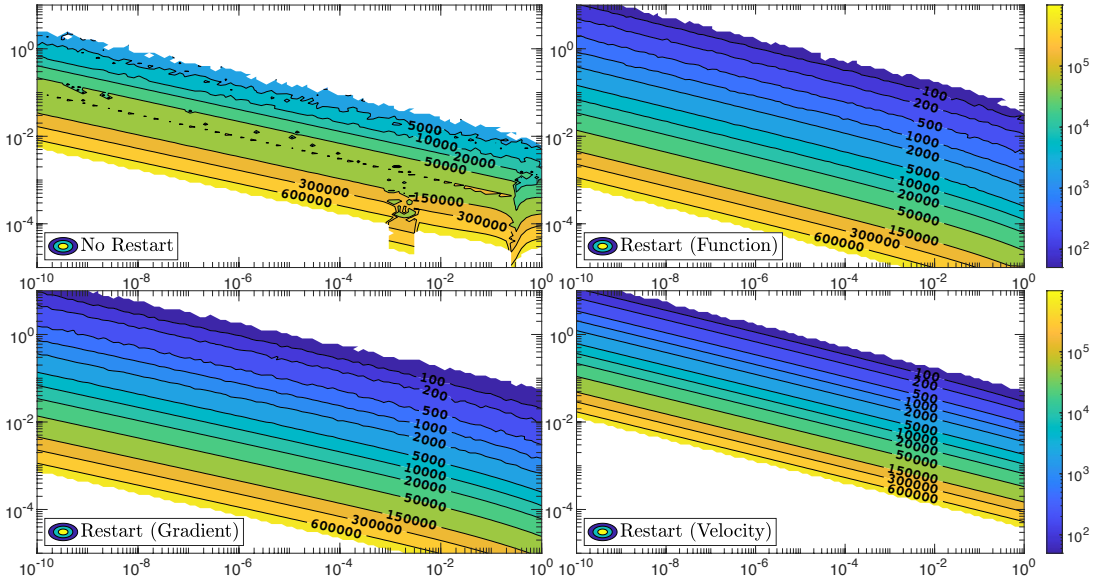

Figure 2.: Contour plot of the number of iterations required to achieve convergence with $\delta = 10^{-5}$ in the (C, h)-plane, for $p = \mathring{p} = 4$ PolyHTVI applied to Problem 2.

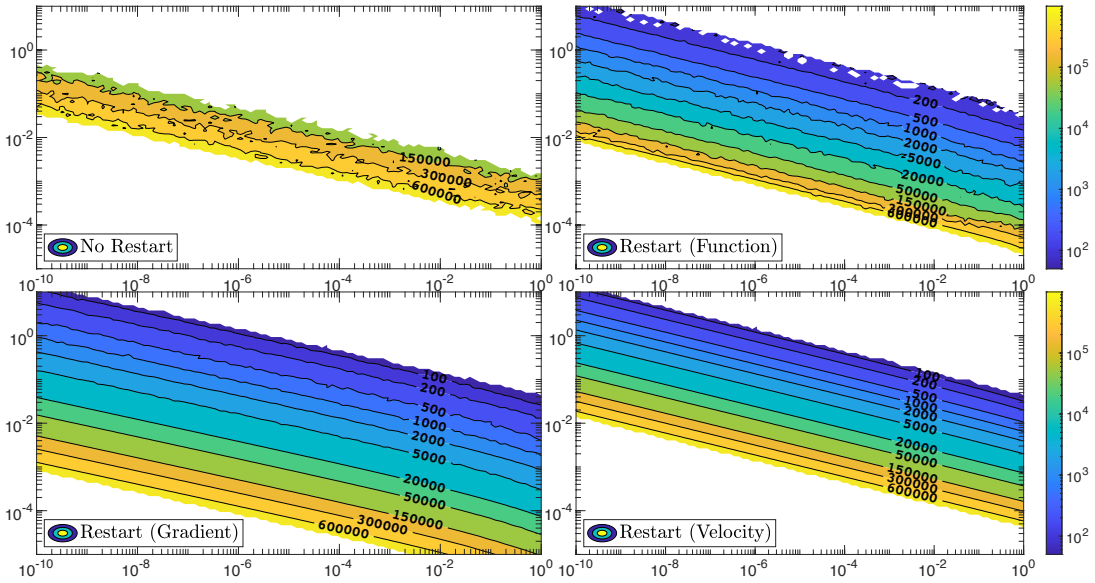

Figure 3.: Contour plot of the number of iterations required to achieve convergence with $\delta = 10^{-8}$ in the (C, h)-plane, for $p = \mathring{p} = 4$ PolyHTVI applied to Problem 2.

Figure 2 confirms the earlier observation that restarting can significantly reduce the number of iterations needed to converge, and we can also see that the restarted versions of the algorithm are more robust, since the regions of fast convergence are larger than for the standard algorithm. As a result, it is easier to tune the restarted algorithms to achieve fast convergence. Note as well from Figure 3 that a restarting scheme can significantly improve the stability of the algorithms: as the convergence criteria are made stricter going from Figure 2 to Figure 3, the regions of fast convergence have not shrunk as dramatically for the restarted algorithms as for the standard version. Given a converging (C, h) pair for a restarted algorithm in Figure 2, the restarted algorithm usually remains convergent for that (C, h) pair with the stricter criteria

in Figure 2 with a slightly increased number of iterations required. This is not true for the standard algorithm where the increase in number of iterations is much more significant, and there is a larger region of initially convergent (C, h) pairs where the standard algorithm diverges when the stricter convergence criteria is imposed.

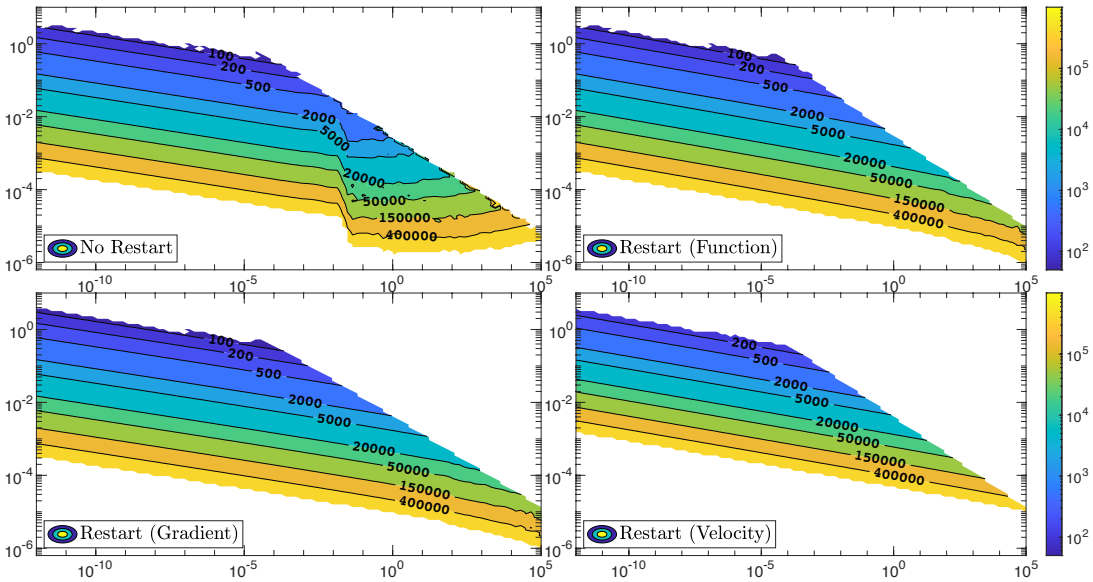

Figure 4.: Contour plot of the number of iterations required to achieve convergence in the (C, h)-plane, for the $p = \mathring{p} = 8$ PolyHTVI algorithm applied to Problem 1.

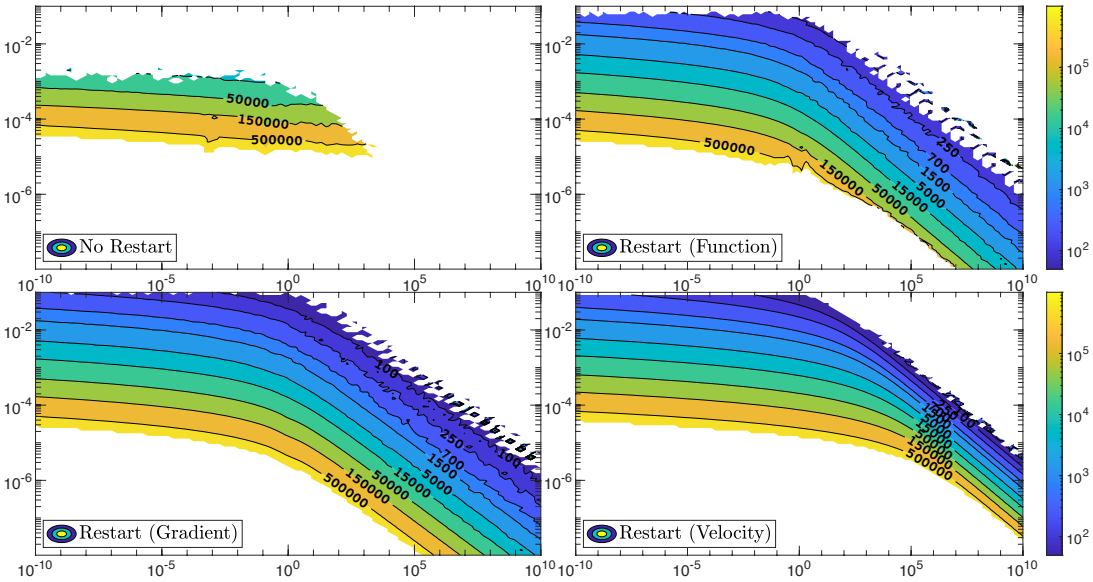

Figure 5.: Contour plot of the number of iterations required to achieve convergence with $\delta = 10^{-5}$ in the (C, h)-plane, for $\eta = \mathring{\eta} = 1$ ExpoSLC applied to Problem 2.

As was observed earlier in Figure 1, we can see from Figure 4 that momentum restarting does not lead to significant improvements in computational efficiency for Problem 1, but also does not penalize computational efficiency in that case. From Figure 4, we see that this observation extends to robustness and stability. since all the different versions of the algorithm share similar convergence regions given the same parameter values and convergence criteria.

All the observations made so far also extend to the other Bregman subfamilies of dynamics and other algorithms, as can be seen, for instance, in Figure 5 for the ExpoSLC algorithm, where momentum restarting leads to significant gains in computational efficiency, robustness and stability. Table 1 provides additional data supporting the significant gain in efficiency that can be achieved using momentum restarting.

Algorithm	**Problem**	δ	**No Restart**	**Function Scheme**	**Gradient Scheme**	**Velocity Scheme**
PolyHTVI	Problem 1	10^{-12}	52	52	52	108
PolyHTVI	Problem 2	10^{-5}	621	39	23	34
PolyHTVI	Problem 2	10^{-8}	15994	80	51	57
PolyHTVI	Problem 3	10^{-8}	4121	60	11	16
PolyHTVI	Problem 4	10^{-8}	14723	60	12	14
PolyHTVI	Problem 5	10^{-5}	3917	104	33	38
ExpoSLC	Problem 1	10^{-12}	75	68	64	155
ExpoSLC	Problem 2	10^{-5}	3929	50	20	21
ExpoSLC	Problem 2	10^{-8}	204598	91	27	32
ExpoSLC	Problem 3	10^{-8}	17081	66	15	18
ExpoSLC	Problem 4	10^{-8}	58028	72	10	12
ExpoSLC	Problem 5	10^{-5}	21440	54	27	41

Table 1.: Comparison of the fastest convergence achieved by the standard algorithms and the restarting schemes on various problems with different tolerances δ (displayed in terms of number of iterations required to achieve the termination criteria).

Overall, all the numerical experiments conducted in this section unequivocally support the use of momentum restarting in the algorithms for accelerated optimization, and it can be seen from Figures 2, 3, 4, and 5 and Table 1 that the gradient based restarting scheme consistently outperforms the other two restarting schemes in terms of computational efficiency, robustness and stability. Unless stated otherwise, we will now always use momentum restarting based on the gradient scheme in the remainder of this paper, without explicitly stating it every time.

5.2. The Effect of the Parameter C

The parameter C in the polynomial and exponential subfamilies of Bregman dynamics, presented in Sections 3.2 and 3.3, can sometimes provide a simple way to control the oscillatory behavior of the second-order differential equation. From the point of view of perturbation theory, the polynomial and exponential Bregman Euler–Lagrange equations (16) and (20),

$$\ddot{q}(t) + \frac{p+1}{t}\dot{q}(t) + Cp^2 t^{p-2}\nabla f(q(t)) = 0, \quad \text{and} \quad \ddot{q}(t) + \eta\dot{q} + C\eta^2 e^{\eta t}\nabla f(q(t)) = 0, \tag{46}$$

can be thought of as perturbations of the simpler differential equations,

$$\ddot{u}(t) + \frac{p+1}{t}\dot{u}(t) = 0, \qquad \text{and} \qquad \ddot{v}(t) + \eta\dot{v} = 0. \tag{47}$$

The solutions to these two unperturbed equations are given by

$$u(t) = \left(k_1 t^{-p} + k_2\right)\mathbb{1}, \qquad \text{and} \qquad v(t) = \left(k_3 e^{-\eta t} + k_4\right)\mathbb{1}, \tag{48}$$

for some constants k_1, k_2, k_3, k_4 depending on the initial conditions. They are non-oscillatory, and converge monotonically to a constant vector at the respective

rates of $\mathcal{O}(t^{-p})$ and $\mathcal{O}(e^{-\eta t})$. We can thus think of the terms $C\eta^2 e^{\eta t}\nabla f(q(t))$ and $C\eta^2 e^{\eta t}\nabla f(q(t))$ as perturbations steering the dynamical system towards the minimizer of the objective function f, in an oscillatory fashion. The parameter C, which appears in front of these two perturbation terms, should therefore be chosen, in theory, to be small enough to control the oscillations but also large enough to guide the dynamical system towards the minimizer of the objective function. The situation is similar in the ExpoToPoly and PolyToExpo subfamilies of Bregman dynamics.

This perturbation theoretic point of view and the numerical results which will be presented in this section suggest that the parameter C can play a very important role reducing the effect of oscillations and improving the performance of optimization algorithms. The benefits that tuning the parameter C can provide have not been sufficiently explored in the literature exploiting the variational framework for accelerated optimization (in [16, 21, 32, 47, 84] for instance), and the resulting dynamical systems were highly-oscillatory and thus required smaller time-steps for their discretizations. As a consequence, the resulting optimization algorithms might not be as competitive as they could have been. Note as well that the limiting continuous differential equation for Nesterov's Accelerated Gradient introduced in [77] can be thought of as the $p = 2$ polynomial Bregman dynamics with $C = 1/4$, which results in the highly oscillatory behavior observed in the continuous dynamics associated to most objective functions, and in the numerous discretizations of these dynamics which can be found in the literature. This observation also extends to the Riemannian manifold generalization of this variational framework for accelerated optimization [31], where the constant C might not have been optimally tuned in practice (in [28–31, 52, 78] for instance).

As a first example, Figures 6 and 7 display the changes in the polynomial and exponential Bregman dynamics for Problem 1 as the parameter C is decreased. The oscillations are clearly neutralized in the continuous Bregman dynamics as C decreases. Although the convergence happens later in time for lower values of C, this is usually not an issue since the neutralization of the oscillations allows for larger time-steps when discretizing, as can be seen in Figure 8. This could also be seen in the 'No Restart' contour plots presented in Figures 2, 3, 4, and 5, where lower values of C allowed for larger time-steps h. Unfortunately, this behavior as C is decreased does not seem to be universal, as can be seen from Figure 9 for Problem 4.

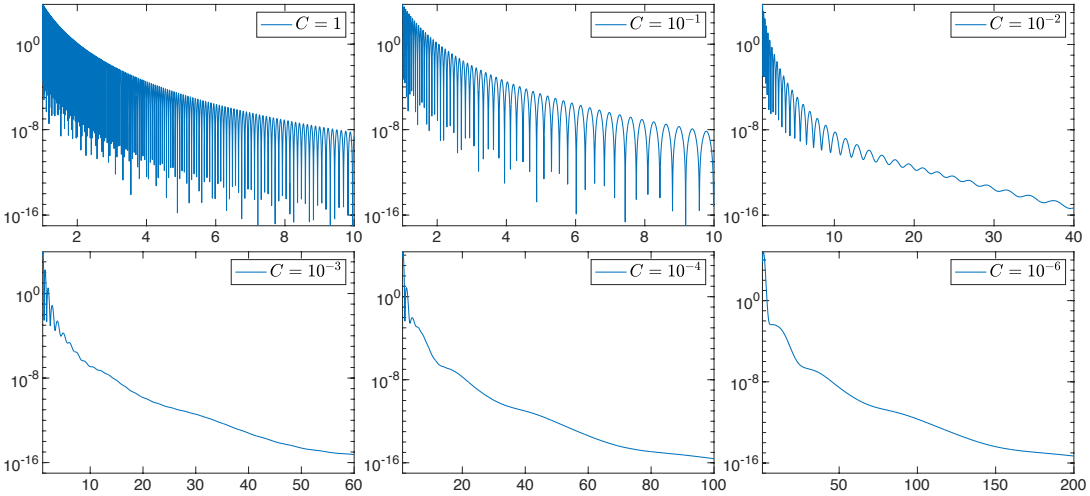

Figure 6.: Error as a function of time t along the $p = \mathring{p} = 6$ polynomial Bregman dynamics for Problem 1, with different values of the constant C.

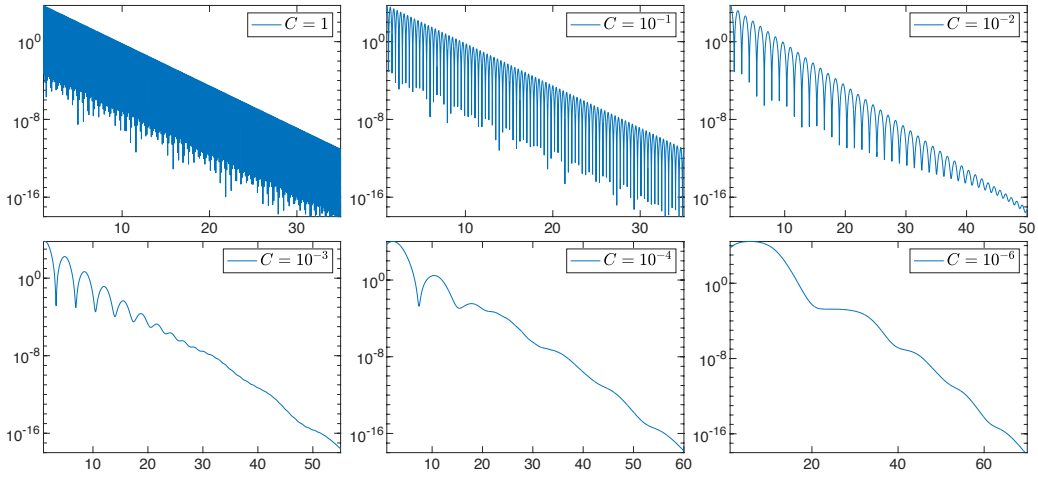

Figure 7.: Error vs t along $\eta = \mathring{\eta} = 0.5$ exponential Bregman dynamics for Problem 1.

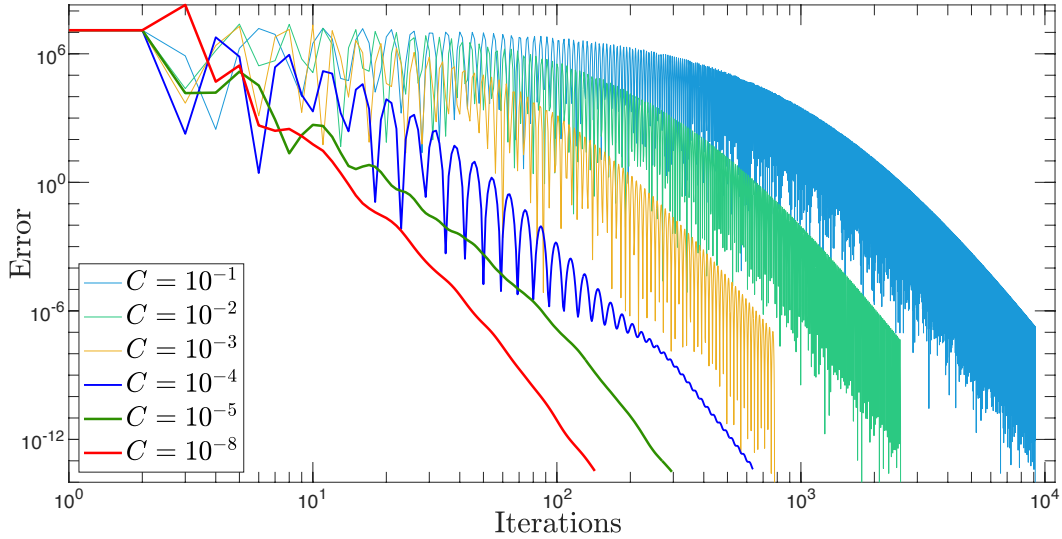

Figure 8.: Discretization of the polynomial Bregman dynamics using PolyHTVI with different values of C for Problem 1 (without momentum restarting).

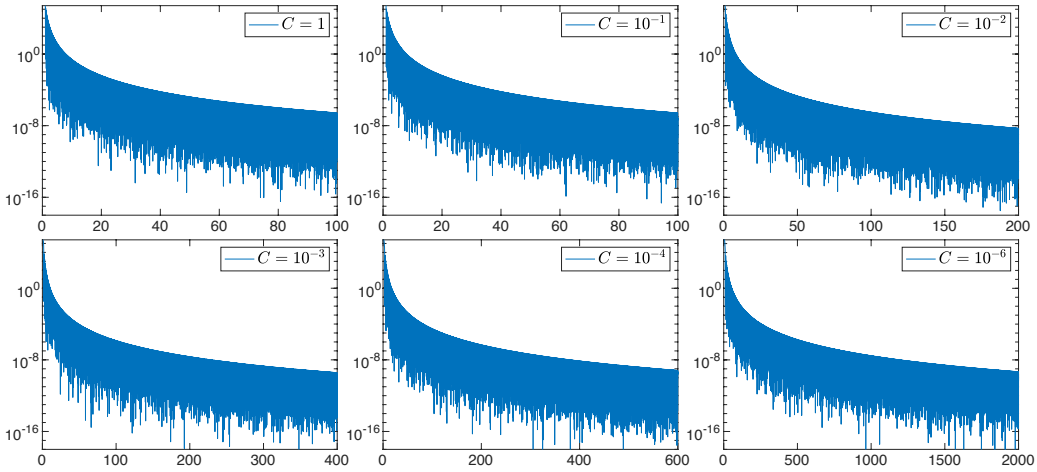

Figure 9.: Error vs. t along $p = \mathring{p} = 6$ polynomial Bregman dynamics for Problem 4.

Further numerical experiments, presented in Supplementary Material 4, show that the regions of optimal convergence are problem-dependent and as a result we cannot find a single value of C which will achieve almost-optimal performance on all problems. However, in these numerical experiments, the convergence regions in the (C, h)-plane were left almost unchanged as the dimension of the problem was increased. This observation can improve significantly the process of tuning the optimization algorithm for high-dimensional problems by first tuning the algorithm on a similar low-dimensional problem, which could be particularly helpful for certain machine learning applications.

5.3. Other Approaches to Control Oscillations

There are other possible approaches to control the oscillations in second-order nonlinear differential equations. One such method is Hessian-driven damping [6, 9–11], where the idea is to add a damping term which involves the Hessian of the objective function, $\beta(t)\nabla^2 f(x(t))\dot{x}(t)$, to the differential equation of interest:

$$\ddot{x}(t) + \gamma(t)\dot{x}(t) + \beta(t)\nabla^2 f(x(t))\dot{x}(t) + b(t)\nabla f(x(t)) = 0. \tag{49}$$

The addition of this Hessian-driven damping term appears to neutralize the oscillations in the continuous solution to the differential equation. Furthermore, it was shown using Lyapunov analysis that under suitable assumptions, solutions to the modified equation not only satisfied a similar convergence rate to the minimizer as solutions to the original equation, but also benefited from additional convergence properties for the norm of the gradient ∇f. First-order optimization algorithms were also derived by discretizing the modified differential equation, after rewriting $\nabla^2 f(x(t))\dot{x}(t)$ as $\frac{d}{dt}\nabla f(x(t))$. Unfortunately, we cannot derive a simple variational formulation for this modified differential equation, so we cannot easily incorporate Hessian-driven damping into our framework which relies on geometric numerical integration of Lagrangian or Hamiltonian systems.

Another possible approach to control oscillations consists in simplifying the Bregman dynamics using local approximations. For instance, one could integrate local linearizations of the Bregman Hamilton's equations, or start from local quadratic Hamiltonian approximations to the Bregman Hamiltonian, or use a local quadratic model for the objective function. We will not consider these methods here because they can suffer from additional numerical stability issues coming from the approximations at play, and it can be very challenging to design a symplectic integrator which preserves the nice properties of the dynamics across all the different local approximations.

A different approach consists in designing a symplectic integrator which can travel faster along the oscillations via larger time-steps. This may be achievable using Spectral or Galerkin variational integrators [44, 55, 56, 60], which rely on a choice of basis functions that span a good approximation space for the Bregman dynamics (for instance, simulations of the polynomial Bregman dynamics suggest that the error usually follows a trajectory which can be well-approximated using functions of the form $t^{-\gamma}\cos(\alpha t^{\beta})$ or $t^{-\gamma}\sin(\alpha t^{\beta})$, where γ is the decay rate, α tunes the frequency of oscillations, and $\beta \in (0, 1)$ characterizes the slowing down of the oscillation frequency). Due to the oscillatory nature of the dynamical system, it might also be advantageous to use Filon-type [37] or Levin-type [57, 58] quadrature rules in the construction of the integrators, since they are designed specifically for highly oscillatory integrals (see [24] for a thorough presentation).

Another possibility involves averaging techniques [34, 73, 74, 83]. The extended Bregman Hamiltonians or Lagrangians can be split as

$$H(\bar{q},\bar{r}) = H^A(\bar{q},\bar{r}) + CH^B(\bar{q}), \quad \text{or} \quad L(\bar{q},\bar{q}') = L^A(\bar{q},\bar{q}') + CL^B(\bar{q}), \tag{50}$$

where the A-component is dominating and can be solved exactly (or efficiently approximated with high accuracy), and the B-component generates small perturbations affecting the overall dynamics. One can then hope to integrate the dominating dynamics very accurately with larger time-steps and incorporate the influence of the small perturbations by averaging them out. Unfortunately, although this approach seemed to neutralize the oscillations in the solution in practice, it did not allow the use of larger time-steps, and the resulting algorithm was actually less competitive and robust because of the implicit nature of the update for the momentum r.

6. Choosing and Tuning a Geometric Integrator

6.1. Time-Adaptivity in the Momentum Restarted Algorithms

In [28–32], numerical experiments with the polynomial Bregman dynamics suggested that time-adaptivity could result in significantly faster optimization algorithms. These experiments were however carried with the standard versions of the algorithms. Although the use of time-adaptivity allows for a larger family of algorithms from which one might be able to extract a more efficient algorithm than without time-adaptivity, numerical experiments presented in Supplementary Material 5 suggest that with momentum restarting, the benefits time-adaptivity may provide are very limited and are not worth the computational effort of tuning one additional parameter. For this reason, we will now discard time-adaptivity, and focus on the non-adaptive approaches. In particular, this allows to discard the ExpoToPoly and PolyToExpo Bregman subfamilies and to focus on the $p = \mathring{p}$ polynomial and $\eta = \mathring{\eta}$ exponential Bregman subfamilies.

6.2. Comparison of Integrators

Numerical experiments presented in Supplementary Material 7 showed that all the geometric integrators perform with very small discrepancies. However, if we had to choose an integrator for the Bregman dynamics, the SLC algorithms with momentum restarting seem to be the slightly better choices. These algorithms will be referred to as PolySLC-R and ExpoSLC-R and are given explicitly in Supplementary Material 7.

6.3. Tuning the Algorithms

In an effort to reduce the number of parameters needing tuning in practice, we investigated how the algorithms perform as the parameters C, h, p, η are varied in Supplementary Material 8. We saw that tuning the values of p or η carefully might not be very helpful and necessary in practice, so we can set and fix their value to $p = 6$ and $\eta = 0.01$. Further experiments showed that there is no universally optimal value of C or h. However, $(C, h) = (0.1, 0.01)$ in the polynomial case and $(C, h) = (1, 4)$ in the exponential case seem to work well for most problems considered and can be used as default values. In conclusion, one would typically only have to try a few logarithmically-spaced values of C in $[10^{-5}, 10^{5}]$ and then adjust h for optimal convergence.

7. Temporal Looping to Improve Numerical Stability

There is an important caveat to the promising performance observed for the optimization algorithms constructed in this paper. The evolution of the variables $q, \mathfrak{q}$ and r associated with the exponential Poincaré Hamiltonian,

$$\bar{H}^{\eta}(\bar{q}, \bar{r}) = \frac{\eta}{2e^{\eta \mathfrak{q}}}\langle r, r\rangle + C\eta e^{2\eta \mathfrak{q}} f(q) + \mathfrak{r}, \tag{51}$$

is guided by Hamilton's equations,

$$\dot{q} = \eta e^{-\eta \mathfrak{q}} r, \qquad \dot{r} = -C\eta e^{2\eta \mathfrak{q}} \nabla f(q), \qquad \dot{\mathfrak{q}} = 1. \tag{52}$$

From these equations of motion, we can see that the time variable $\mathfrak{q}$ grows linearly without bound, and as a result quantities like $e^{\eta \mathfrak{q}}$ grow exponentially without bound. More precisely, looking at the updates of the ExpoSLC algorithm,

$$r \leftarrow r - C\eta h e^{2\eta \mathfrak{q}} \nabla f(q), \qquad \mathfrak{q} \leftarrow \mathfrak{q} + h, \qquad q \leftarrow q + \Delta q = q + \eta h e^{-\eta\left(\mathfrak{q} + \frac{h}{2}\right)} r, \tag{53}$$

we have at every iteration that the new update is given by

$$(\Delta q)_{new} \leftarrow A(\Delta q)_{previous} + Be^{\eta \mathfrak{q}} \nabla f(q), \tag{54}$$

for some constants A and B.

If we could perform all the operations exactly, the gradient ∇f would converge to 0 with arbitrary precision and neutralize the unbounded growth of $e^{\eta \mathfrak{q}}$, and the quantity $Be^{\eta \mathfrak{q}} \nabla f(q)$ would remain very small. However, in practice, we can only perform operations with finite precision in floating-point arithmetic. As a result, ∇f only decays to 0 up to machine precision while $e^{\eta \mathfrak{q}}$ grows without bound. Eventually, $Be^{\eta \mathfrak{q}} \nabla f(q)$ becomes large again and the position variable q moves away from the equilibrium it found near its optimal value. Something analogous happens in the polynomial family of Bregman dynamics, except that the unbounded growing exponential is replaced by an unbounded growing polynomial. This numerical instability phenomenon is illustrated in Figure 10 which displays the evolution of the error $|f(x_k) - f(x^*)|$ when the SLC algorithms are applied to Problem 2. We see that both algorithms first achieve convergence to machine precision, stay at the minimizer for a few hundred iterations, and finally are expelled away from the minimizer due to numerical instability.

In all our numerical experiments so far, the algorithms stopped when they reached a desired convergence criterion, so we did not observe this numerical instability issue as it happens only after convergence is achieved. However, in practice, optimization algorithms are often terminated after a specified number of iterations instead of a specified convergence criterion. Thus, we need a strategy to avoid this numerical instability phenomenon. Since the numerical instability results from the limitation imposed by machine precision on accurately representing the decay to 0 of $\nabla f(q)$ while the term $e^{\eta \mathfrak{q}}$ grows without bound, it is natural to try to restrict the growth of the term $e^{\eta \mathfrak{q}}$, by restricting the growth of $\mathfrak{q}$ (and similarly in the polynomial case).

One possibility is to reset time whenever a certain numerical instability criterion is met, via $\mathfrak{q} \leftarrow \beta \mathfrak{q}$ for some $\beta \in (0, 1)$. A larger β is preferable to keep enough momentum in case convergence to the minimizer was only suboptimal when the numerical instability criterion was met, or if the algorithm is used in an online fashion or with a stochastic or

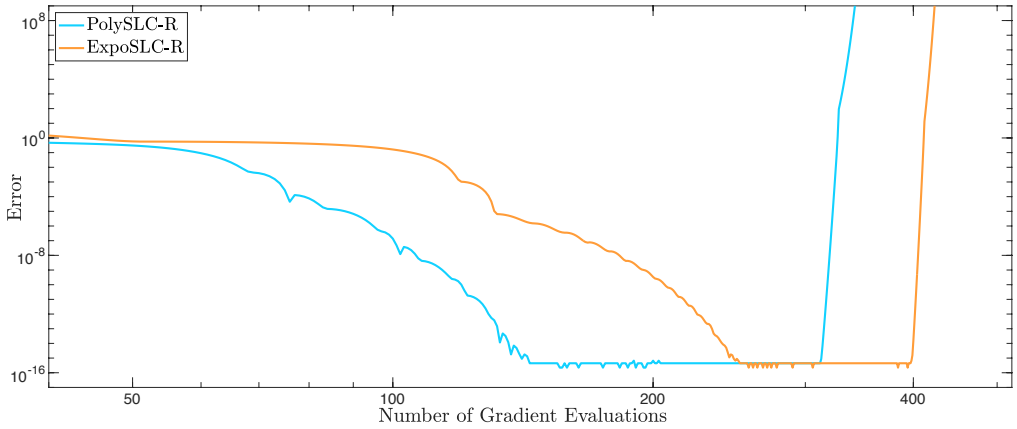

Figure 10.: The PolySLC-R and ExpoSLC-R algorithms applied to Problem 2.

mini-batch approach. It is also preferable to avoid values of β very close to 1, since the algorithm would then always remain close to numerical instability, and could possibly become unstable if the criterion is not chosen very carefully. In practice, taking β between 0.6 and 0.95 works well, by ensuring that a reasonable amount of momentum is kept while avoiding the numerical instability region. Alternatively, one could reset time via $\mathfrak{q} \leftarrow \mathfrak{q} - \nu h$ for some $\nu > 1$. A smaller ν is preferable to retain momentum, while ν should not be too close to 1 to avoid numerical instability. In practice, we will reset time via $\mathfrak{q} \leftarrow \max(\epsilon, \beta\mathfrak{q})$ or $\mathfrak{q} \leftarrow \max(\epsilon, \mathfrak{q} - \nu h)$, where ϵ is a small positive number, to avoid very small or negative values of time $\mathfrak{q}$. This phenomenon where the time variable $\mathfrak{q}$ is stuck in a loop by resetting $\mathfrak{q} \leftarrow \beta\mathfrak{q}$ or $\mathfrak{q} \leftarrow \mathfrak{q} - \nu h$ whenever numerical instability is near will be referred to as **Temporal Looping**.

Improving the ExpoSLC-R algorithm via temporal looping yields Algorithm 1:

Algorithm 1: ExpoSLC-RTL: **S**ymmetric **L**eapfrog **C**omposition for the **Expo**nential Bregman dynamics, with **R**estarting and **T**emporal **L**ooping

Input: An objective function $f : \mathbb{R}^d \to \mathbb{R}$. An initial guess $q \in \mathbb{R}^d$.
Parameters $C, h, p > 0$, $\beta \in (0, 1)$ or $\nu > 1$.

$\epsilon \leftarrow 0.001, \quad \mathfrak{q} \leftarrow 1, \quad G \leftarrow \nabla f(q), \quad r \leftarrow -\frac{1}{2} C\eta h e^{2\eta\mathfrak{q}} G$
while *convergence criteria are not met* **do**
  $\Delta q \leftarrow \eta h e^{-\eta(\mathfrak{q}+\frac{h}{2})} r$
  $q \leftarrow q + \Delta q$
  $G \leftarrow \nabla f(q)$
  if $G^\top \Delta q > 0$ **then** restart momentum: $r \leftarrow 0$
  if *numerical instability criterion is met* **then** $\mathfrak{q} \leftarrow \max(\epsilon, \beta\mathfrak{q})$ or $\mathfrak{q} \leftarrow \max(\epsilon, \mathfrak{q} - \nu h)$
  $\mathfrak{q} \leftarrow \mathfrak{q} + h$
  $r \leftarrow r - C\eta h e^{2\eta\mathfrak{q}} G$

In our numerical experiments with ExpoSLC-RTL, we use the instability criterion

$$Ch^2\eta^2 e^{\eta\mathfrak{q}} \|G\| > e^{-\eta h} \|\Delta q\|, \tag{55}$$

to reset time. This criterion roughly ensures that $|B| e^{\eta\mathfrak{q}} \|\nabla f(q)\| < |A| \|(\Delta q)_{previous}\|$ in equation (54), so that $(\Delta q)_{new}$ is not significantly larger in norm than $(\Delta q)_{previous}$.

Improving the PolySLC-R algorithm via temporal looping yields Algorithm 2:

Algorithm 2: PolySLC-RTL: **S**ymmetric **L**eapfrog **C**omposition for the **Poly**nomial Bregman dynamics, with **R**estarting and **T**emporal **L**ooping

Input: An objective function $f : \mathbb{R}^d \to \mathbb{R}$. An initial guess $q \in \mathbb{R}^d$.
Parameters $C, h, p > 0$, $\beta \in (0, 1)$ or $\nu > 1$.

$\epsilon \leftarrow 0.001, \quad \mathfrak{q} \leftarrow 1, \quad G \leftarrow \nabla f(q), \quad r \leftarrow -\frac{1}{2} Chp\mathfrak{q}^{2p-1} G$
while *convergence criteria are not met* **do**
  $\Delta q \leftarrow hp\left(\mathfrak{q} + \frac{h}{2}\right)^{-p-1} r$
  $q \leftarrow q + \Delta q$
  $G \leftarrow \nabla f(q)$
  if $G^\top \Delta q > 0$ **then** restart momentum: $r \leftarrow 0$
  if *numerical instability criterion is met* **then** $\mathfrak{q} \leftarrow \max(\epsilon, \beta\mathfrak{q})$ or $\mathfrak{q} \leftarrow \max(\epsilon, \mathfrak{q} - \nu h)$
  $\mathfrak{q} \leftarrow \mathfrak{q} + h$
  $r \leftarrow r - Chp\mathfrak{q}^{2p-1} G$

In our numerical experiments, we have chosen the numerical instability criterion

$$Ch^2 p^2 (\mathfrak{q} + h)^{p+1} \|G\| > \mathfrak{q} \|\Delta q\|, \tag{56}$$

which roughly ensures that the new position update is not significantly larger than the previous one.

Figure 11 shows that temporal looping takes care of the numerical instability issue experienced earlier in Figure 10 for Problem 2:

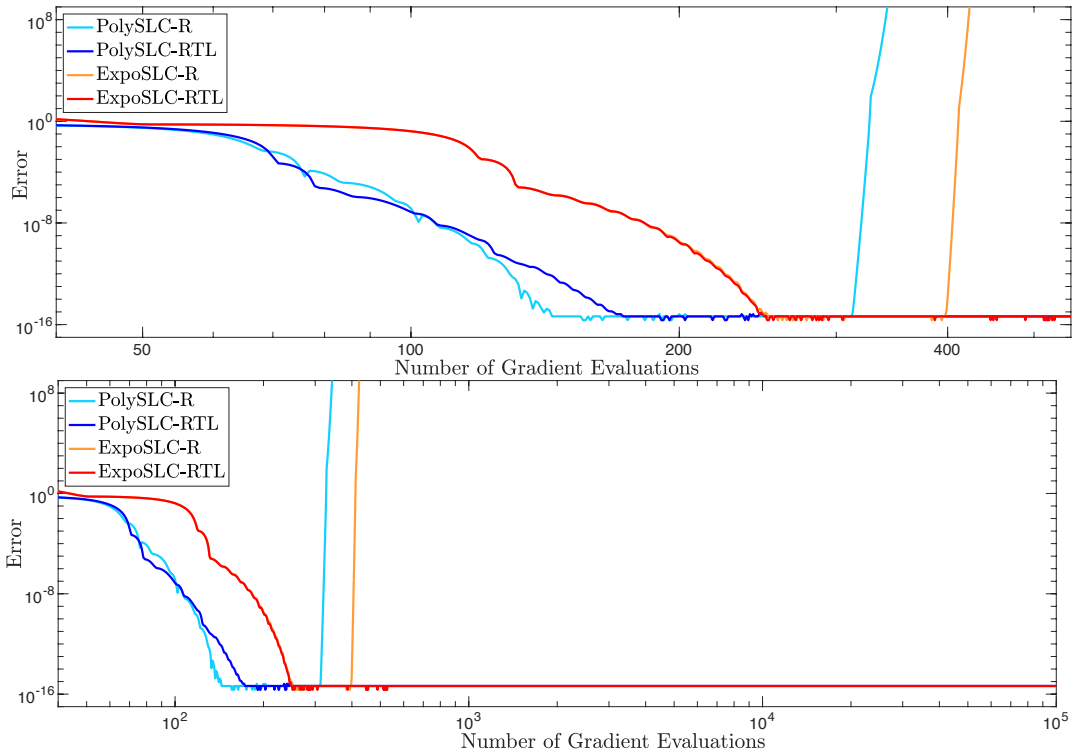

Figure 11.: The effect of temporal looping in PolySLC-R and ExpoSLC-R.

It can be seen from Figures 12 and 13 that temporal looping, with the $\mathfrak{q} \leftarrow \max(\epsilon, \beta\mathfrak{q})$ scheme or $\mathfrak{q} \leftarrow \max(\epsilon, \mathfrak{q} - \nu h)$ scheme, does not negatively affect the performance of the algorithms, although the algorithms with temporal looping might sometimes require a larger number of iterations to achieve convergence for a fixed (C, h)-pair. Indeed the regions of fast convergence might be shifted slightly, but remained at least as large if not larger when using temporal looping.

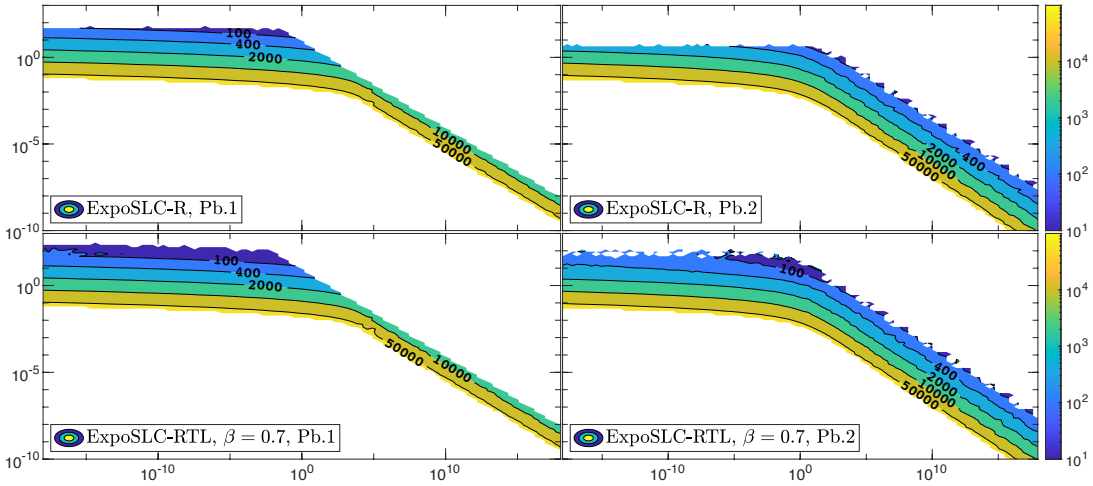

Figure 12.: Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-8}$) in the (C, h)-plane, for the ExpoSLC-R and ExpoSLC-RTL algorithms, when applied with $\eta = 0.01$ to Problems 1 and 2.

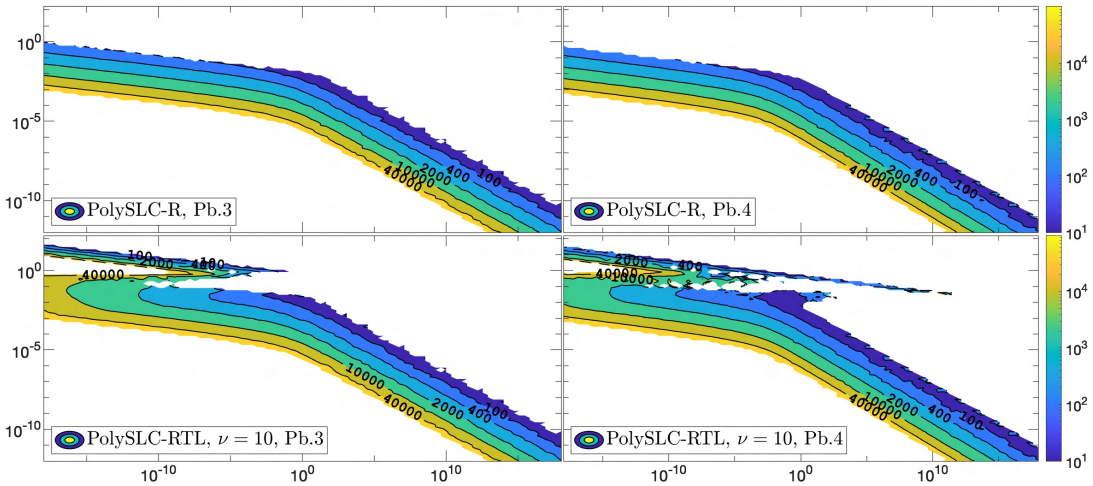

Figure 13.: Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-8}$) in the (C, h)-plane, for the PolySLC-R and PolySLC-RTL algorithms, when applied with $p = 10$ to Problems 3 and 4.

Overall, we have seen that temporal looping can be very helpful to deal with post-convergence numerical instability, and that it does not affect negatively the initial performance of the algorithm. Note that temporal looping could be improved by tuning the parameters β or ν, or by designing a better suited numerical instability criterion.

8. Testing for Machine Learning Applications

We now test our algorithms on more challenging optimization problems for machine learning with a variety of model architectures, loss functions, and applications. For most of the problems considered in this section, the gradients are evaluated in a mini-batch fashion. For reference, we will also solve these optimization problems using gradient descent and the most commonly used optimizer in machine learning, Adam [48]:

ADAM

$$
\begin{aligned}
m_{k+1} &= \beta_1 m_k + (1-\beta_1)\nabla f(x_k) \\
v_{k+1} &= \beta_2 v_k + (1-\beta_2)\nabla f(x_k) \odot \nabla f(x_k) \\
x_{k+1} &= x_k - h\left[(1-\beta_1^{k+1})\left(\sqrt{(1-\beta_2^{k+1})^{-1} v_{k+1}} + \epsilon\right)\right]^{-1} m_{k+1}
\end{aligned}
$$

Here, $u \odot v$ denotes elementwise multiplication, and the update for x_{k+1} is performed elementwise. The variable ϵ present in the updates of the Adam algorithm is there to avoid numerical instability associated with division by 0 (with default value $\epsilon = 10^{-8}$ in [48] and PyTorch). The three parameters of Adam are β_1, β_2 used for computing running averages of gradients, and the learning rate h (with default values $\beta_1 = 0.9$, $\beta_2 = 0.999$, $h = 0.001$ in [48], PyTorch and TensorFlow).

The ExpoSLC-RTL and PolySLC-RTL algorithms have been implemented under the more evocative names **eBrAVO** and **pBrAVO** (**Br**egman **A**ccelerated **V**ariational **O**ptimizer) in Python so that they can be used within the TensorFlow and PyTorch frameworks. These algorithms are available at github.com/vduruiss/AccOpt_via_GNI, and can be called in a similar way as the Adam algorithm in TensorFlow and PyTorch:

```
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)
optimizer = BrAVO_tf.eBravo(learning_rate = 1)
```

```
optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)
optimizer = BrAVO_torch.eBravo(model.parameters(), lr = 1)
```

The purpose of this section is not to do a very careful computational comparison of the BrAVO algorithms with commonly used optimization algorithms in machine learning but rather to show that the BrAVO algorithms can be used conveniently within the PyTorch and TensorFlow frameworks for numerous concrete machine learning applications, and that they might be worth considering and improving in the future. A very careful computational comparison of optimization algorithms for machine learning is a much more ambitious goal which is beyond the scope of this paper, and would be more meaningful once the implementation of the BrAVO algorithms within the PyTorch and TensorFlow frameworks has been highly-optimized.

We have first tested the performance of our algorithms with automatic differentiation on instances of the Binary Classification Problem 6 and the Fermat–Weber Location Problem 7. Figure 14 shows the evolution of the loss function (39) when formulating a model separating blue and red regions of 2-dimensional space using a line based on the displayed 500 randomly generated points. Figure 15 shows the evolution of the loss function (40) when solving the Fermat–Weber Location Problem 7 with 5000 randomly generated vectors in $\mathbb{R}^{1000}$ and 5000 randomly generated corresponding scalar weights. We can see from Figures 14 and 15 that our algorithms solve the binary classification and location problems with an accuracy and efficiency comparable to those of the Adam and standard gradient descent (SGD) algorithms implemented in TensorFlow.

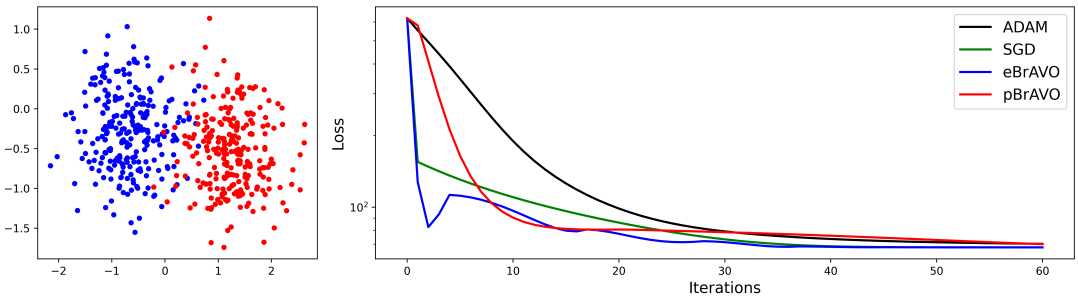

Figure 14.: Comparison of algorithms applied to a Binary Classification Problem 6.

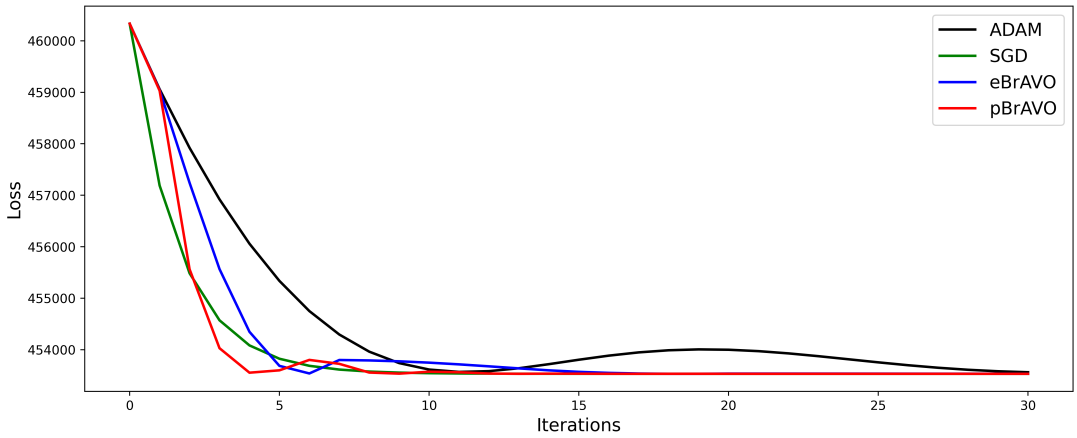

Figure 15.: Comparison of algorithms applied to a Location Problem 7.

Next, we tested our algorithm on the popular multi-label image classification problem based on the Fashion–MNIST dataset [85]: *'Fashion–MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes (t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot)'*. We use `nn.CrossEntropyLoss()` as the loss function, and the following neural network architecture in PyTorch as our classification model:

```
Layer (type)    Output Shape      Parameters
=================================================================
dense (Dense)   [-1, 784]         0
dense_1 (Dense) [-1, 64]          50,240
dense_2 (Dense) [-1, 64]          0
dense_3 (Dense) [-1, 64]          4,160
=================================================================
Total Number of Parameters: 55,050
```

Figure 16 shows that the BrAVO algorithms achieve comparable accuracy and efficiency on the Fashion–MNIST classification problem as the Adam and gradient descent (SGD) algorithms. Note that the momentum restarting scheme and the temporal looping strategy are essential to the good behavior of the algorithms. Indeed, we can see from Figure 17 that without them, the algorithms eventually lose convergence due to numerical instability. Note as well that these strategies can also allow for larger time-steps which usually translates into faster convergence.

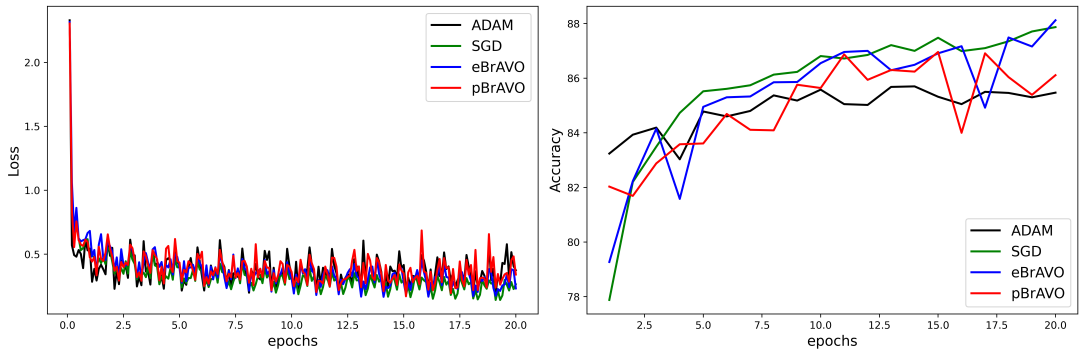

Figure 16.: Evolution of the loss function and accuracy (in %) of Adam, standard gradient descent (SGD) and the BrAVO algorithms, when applied to the Fashion–MNIST multi-label classification problem.

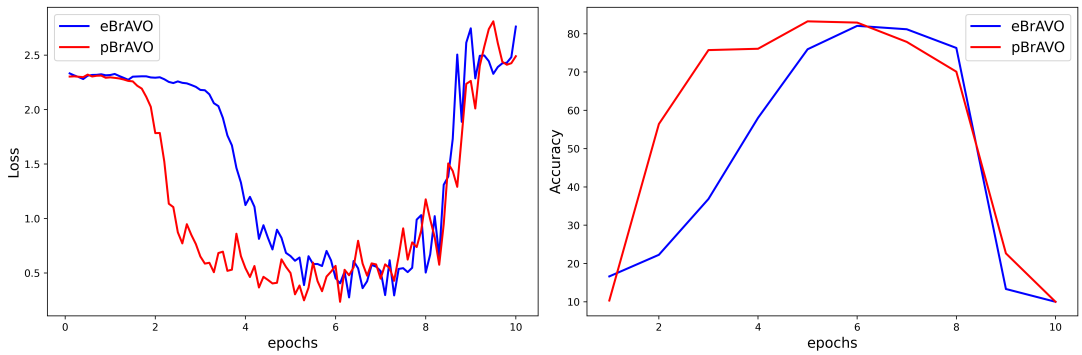

Figure 17.: Convergence and loss of convergence for the BrAVO algorithms without momentum restarting and temporal looping, when applied to the Fashion–MNIST multi-label classification problem.

We then tested our algorithm on another popular multi-label image classification problem based on the CIFAR-10 dataset [49]: *'the CIFAR-10 dataset consists of 60000 32×32 color images in 10 mutually exclusive classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6000 images per class'*. The results are displayed in Figure 18.

We used `nn.CrossEntropyLoss()` as the loss function to minimize and a small Convolutional Neural Network in PyTorch very similar to the LeNet-5 architecture first described in [51]:

```
Layer (type)    Output Shape      Parameters
================================================
Conv2d-1      [-1, 6, 28, 28]     456
Conv2d-2      [-1, 16, 10, 10]    2,416
Linear-3      [-1, 120]           48,120
Linear-4      [-1, 84]            10,164
Linear-5      [-1, 10]            850
================================================
Total Number of Parameters: 62,006
```

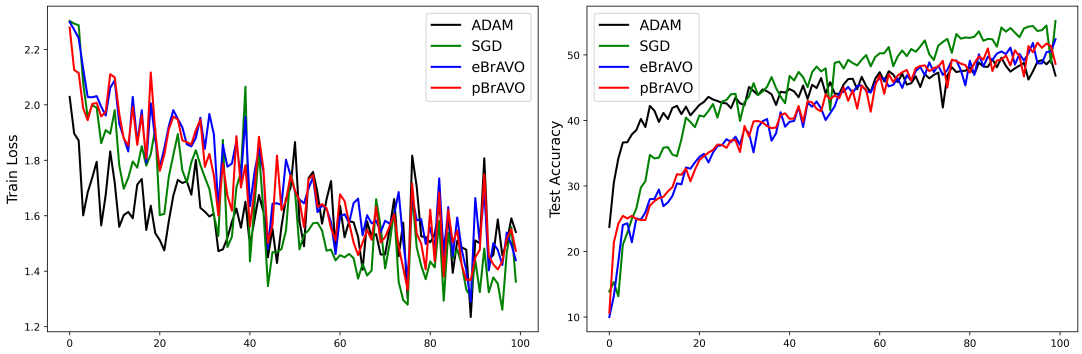

Figure 18.: Evolution over 20 epochs of the loss function and accuracy of various algorithms when applied to the CIFAR–10 multi-label image classification problem.

Let us now consider the Natural Language Processing problem of constructing a multi-label text classifier which can provide suggestions for the most appropriate subject areas for arXiv papers based on their abstracts. The code and architecture used are based on the Keras tutorial [68]. An arXiv paper can belong to multiple categories, so the prediction task can be divided into a series of multiple binary classification problems, and we can use the Binary Cross Entropy loss. We will use the following neural network architecture:

```
model = keras.Sequential()
model.add(layers.Dense(units=256, activation='relu'))
model.add(layers.Dense(units=256, activation='relu'))
model.add(layers.Dense(units=lookup.vocabulary_size(), activation='sigmoid'))
```

The evolution of the loss on the training and validation datasets is displayed in Figure 19. Although the Adam optimizer achieves the smallest loss on the training dataset, the resulting optimized model does not outperform the models generated by the other optimizers on the validation dataset. Its validation loss actually worsens as the epoch number increases (unlike for the other algorithms) which indicates that the optimized model might be suffering from overfitting.

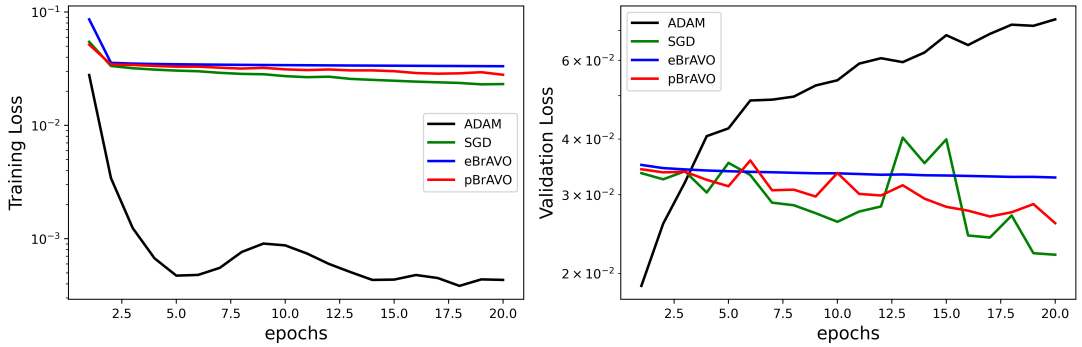

Figure 19.: Evolution of the Binary Cross Entropy loss function on training and validation datasets for several algorithms, when applied to the Natural Language Processing problem of multi-label text classification of arXiv papers.

Next, we consider timeseries forecasting for weather prediction, based on the Keras tutorial [12]. We use the Mean Squared Error (MSE) as the loss function and the following Long Short-Term Memory (LSTM) model (with 5,153 parameters):

```
inputs = layers.Input(shape=(inputs.shape[1], inputs.shape[2]))
lstm_out = layers.LSTM(32)(inputs)
outputs = layers.Dense(1)(lstm_out)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The evolution of the mean squared error on the training and validation sets is displayed in Figure 20. We can see that the four different algorithms generate similar losses on the training and validation datasets.

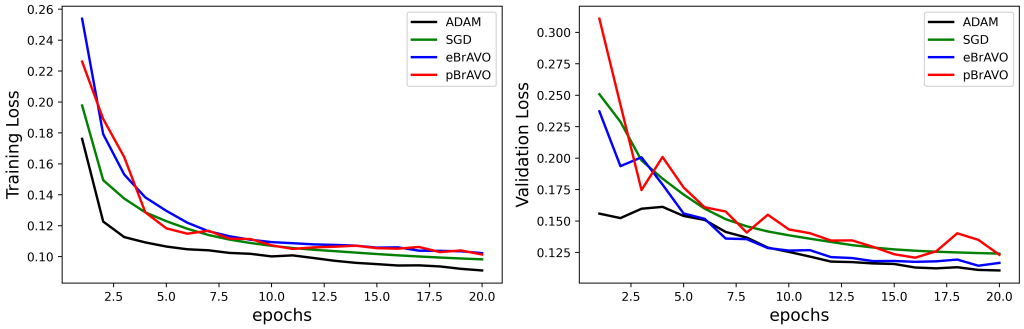

Figure 20.: MSE evolution on training and validation datasets for several algorithms, when used to optimize a LSTM model for timeseries forecasting for weather prediction.

Then, we solved a data fitting problem: given 500 data points from a noisy version of the function $10x|\cos 2x| + 10\exp(-\sin x)$ on the interval $[-2, 2]$, we wish to obtain a model which fits these data points as well as possible. We used the following neural network architecture (with 4,355 parameters) and loss function in TensorFlow:

```
model = keras.Sequential()
model.add(layers.Dense(units = 1, activation = 'linear', input_shape=[1]))
model.add(layers.Dense(units = 64, activation = 'relu'))
model.add(layers.Dense(units = 64, activation = 'relu'))
model.add(layers.Dense(units = 1, activation = 'linear'))
```

The results are displayed in Figures 21 and 22. We see that all the algorithms achieve very small mean squared error, and all generate models, plotted as blue curves in Figure 22, which fit the green data points very well.

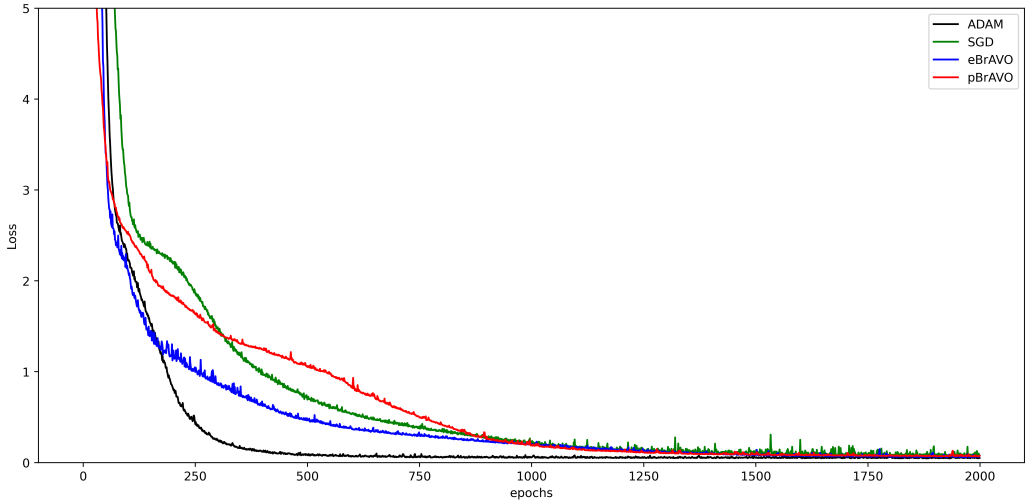

Figure 21.: Evolution of the mean squared error for various algorithms, when applied to the problem of fitting a model to a set of 500 data points.

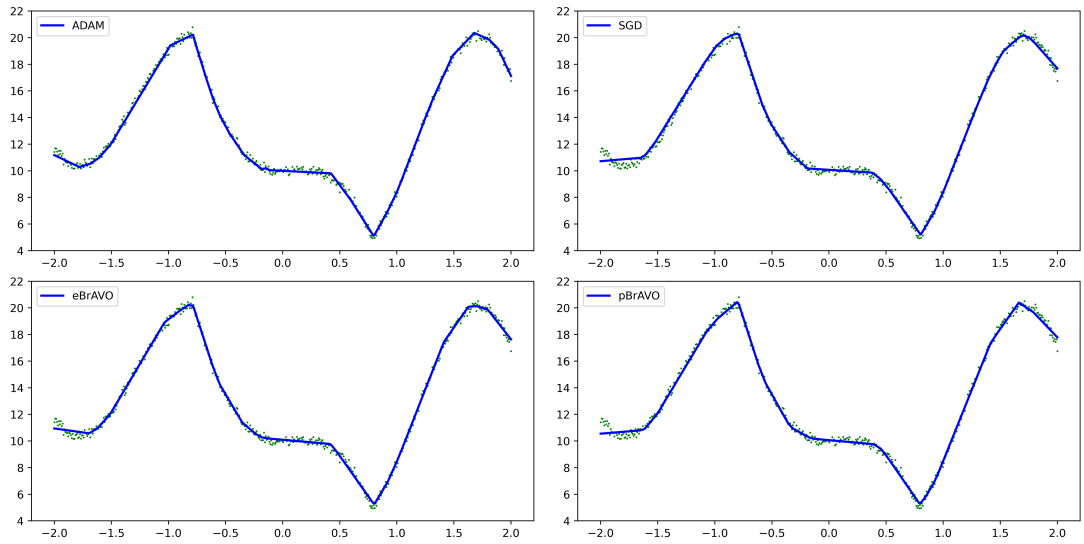

Figure 22.: Models obtained after 2000 epochs using various algorithms to fit the 500 data points displayed in green.

Finally, we test our algorithms for dynamics learning and control on the rotation group $SO(3)$. We consider the same problem as in [27, 33], where we wish to learn the dynamics of a fully-actuated pendulum with dynamics given by $\ddot{\varphi} = -15 \sin \varphi + 3u$, where φ is the angle of the pendulum with respect to its vertically downward position and u is a scalar control input. The data is collected from an OpenAI Gym environment, provided by [91]. We can see from Figure 23 that Adam and the BrAVO algorithms can achieve good training and test losses on this system identification problem using the Hamiltonian-based neural ODE network from [27] (with 231,310 parameters), inspired by [40, 91]. Note that we were unable to tune SGD to obtain a similar performance.

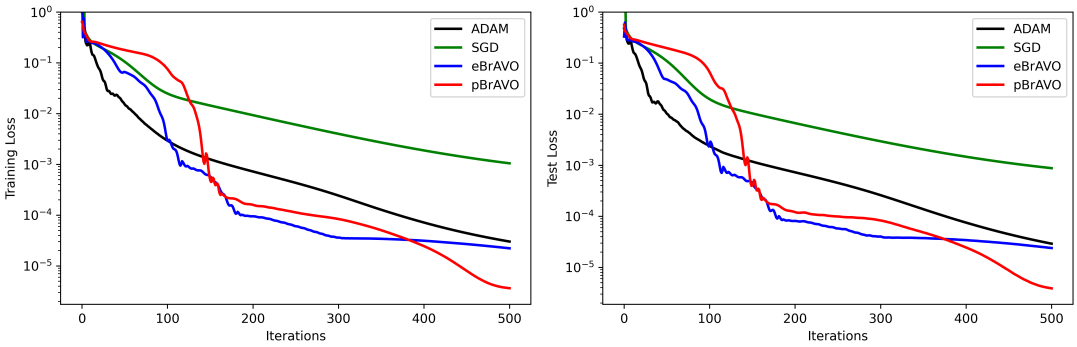

Figure 23.: Evolution of the training and test losses for various algorithms, when learning the 231,310 parameters of a neural ODE network for dynamics learning.

Overall, we have demonstrated here that the BrAVO algorithms can be used conveniently within the PyTorch and TensorFlow frameworks, and that they can perform very well on more challenging optimization problems arising in machine learning applications, with a variety of model architectures, loss functions, and applications. We would like to reiterate that this was the main purpose of this section, and that it is not our intention to make a very careful computational comparison of the BrAVO algorithms with other optimization algorithms that are commonly used by the machine learning community.

A very careful computational comparison of optimization algorithms for machine learning is a much more ambitious goal which is beyond the scope of this paper. Such a comparison would be more meaningful once the current rudimentary implementation of the BrAVO algorithms within the PyTorch and TensorFlow frameworks has been highly-optimized, to take advantage of hardware architectures and highly-optimized PyTorch/TensorFlow operations. Aside from the quality of the implementation, other practical aspects of the algorithm could be investigated and improved further before carrying a careful comparison, for instance by looking into ways to boost the performance of the temporal looping technique or of the momentum restarting scheme.

An important advantage of our methods is that they are derived by discretizing continuous-time dynamical systems. We might be able to derive theoretical results about the algorithm by considering the associated continuous-time dynamical system and the discretization used. Furthermore, by considering the associated continuous-time dynamical system, we may be able to leverage numerous results from the theory of differential equations, dynamical systems, and geometric numerical integration. As a first example, in Section 5.2, we exploited perturbation theory for continuous-time dynamical systems to gain insight into the effect of the parameter C on the performance of the algorithms, which enabled us to improve tuning. As a second example, numerous ideas from the continuous-time theory of dynamical systems have been exploited in [6, 9–11] and in particular the notions of dissipation, of visous and Hessian-driven damping, and of inertia, in second-order differential equations. As a last example, the notion of momentum itself is better understood as a physical property of a continuous-time dynamical system, and we can also gain a lot of insight into the mechanism allowing the accelerated convergence towards the minimizer by considering these dynamical systems. There might be many other ways in which the performance of our algorithms can be improved by leveraging the associated continuous-time dynamical system.

Conclusion and Future Directions

In this paper, we have discussed practical considerations which can significantly boost the computational performance and ease the tuning of symplectic accelerated optimization algorithms that are constructed by integrating Lagrangian and Hamiltonian systems coming from the variational framework for optimization introduced in [84].

We showed that momentum restarting schemes can lead to a significant gain in computational efficiency and robustness by reducing the undesirable effect of oscillations, and that a temporal looping strategy helps to avoid instability issues caused by numerical precision, and does not impair the computational performance of the algorithms in general. We also observed that time-adaptivity and the choice of symplectic integrator hardly make a difference once a momentum restarting scheme is incorporated in the optimization algorithms. This observation, along with other numerical experiments designed to study the effects of the different parameters, has provided insights that allowed to inform and ease the tuning process by simplifying the algorithms and by reducing the number of parameters to tune.

Overall, we have designed symplectic accelerated optimization algorithms whose computational efficiency and stability have been improved using temporal looping and momentum restarting, and which are now more user-friendly. We tested these algorithms on machine learning optimization problems with numerous different model architectures, loss functions, and applications, and saw that they can achieve very good results when tuned properly.

Preliminary experiments suggest that the benefits of momentum restarting and temporal looping uncovered in this paper extend to the Riemannian manifold framework for accelerated optimization introduced in [31]. We intend to explore that direction to improve the computational efficiency and stability of symplectic accelerated optimization algorithms on Riemannian manifolds.

It would be nice to have further theoretical guarantees about the convergence of the discrete algorithm. However, this could be very difficult to obtain because momentum methods lack contraction, are nondescending, and are highly oscillatory [67]. While it is hoped that the continuous analysis will eventually guide the convergence analysis of the discrete-time algorithms, this does not appear to be a straightforward exercise, as one would first need to reconcile the arbitrarily fast rates of convergence of the continuous-time trajectories with Nesterov's barrier theorem of $\mathcal{O}(1/k^2)$ for discrete-time algorithms. We note however that some theoretical guarantees for certain integrators applied to the polynomial subfamily were obtained in the case where $p > 2$ in [90], although this was already a very complicated task achieved under additional assumptions on the objective function and its derivatives. In the future, we intend to try to build upon the results of [90] to derive more general theoretical guarantees for our discrete algorithms, and see how momentum restarting and temporal looping affect those guarantees.

The temporal looping technique could also be improved by designing different numerical instability criteria. Instead of temporal looping strategies, one could also try to implement very popular techniques in machine learning such as decaying learning rates via a learning rate scheduler, or to progressively increase the batch size [76], or a combination of these different approaches.

The current implementation of the algorithms within the PyTorch and TensorFlow frameworks is rather rudimentary, and can certainly be improved to reduce computational time by taking advantage of hardware architectures and highly-optimized PyTorch/TensorFlow operations. With the same objective in mind, one could also replace the gradient scheme for momentum restarting by the function scheme if the latter can be implemented more efficiently.

Once the algorithms have been improved further, possibly leveraging the theory of continuous-time dynamical systems, and once the implementation of the algorithms has been highly-optimized, it would be very interesting to perform a very careful computational comparison with other popular algorithms on many different types of problems to see whether the BrAVO algorithms can outperform the state-of-the-art algorithms on certain classes of machine learning problems.

Funding

The authors were supported in part by NSF under grants DMS-1411792, DMS-1345013, DMS-1813635, CCF-2112665, by AFOSR under grant FA9550-18-1-0288, and by the DoD under grant HQ00342010023 (Newton Award for Transformative Ideas during the COVID-19 Pandemic).

Disclosure statement

The authors report there are no competing interests to declare.

Data availability statement

Implementations of the optimization algorithms in MATLAB, Julia and Python (with TensorFlow and PyTorch) can be found at github.com/vduruiss/AccOpt_via_GNI.

References

[1] K. Ahn and S. Sra. From Nesterov's estimate sequence to Riemannian acceleration. In *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 84–118. PMLR, 09–12 Jul 2020.

[2] C. D. Alecsa and S. C. László. Tikhonov regularization of a perturbed heavy ball system with vanishing damping. *SIAM Journal on Optimization*, 31(4):2921–2954, 2021. .

[3] F. Alimisis, A. Orvieto, G. Bécigneul, and A. Lucchi. Practical accelerated optimization on Riemannian manifolds. 2020. URL https://arxiv.org/abs/2002.04144.

[4] F. Alimisis, A. Orvieto, G. Bécigneul, and A. Lucchi. A continuous-time perspective for modeling acceleration in Riemannian optimization. In *Proceedings of the 23rd International AISTATS Conference*, volume 108 of *PMLR*, pages 1297–1307, 2020.

[5] F. Alimisis, A. Orvieto, G. Bécigneul, and A. Lucchi. Momentum improves optimization on Riemannian manifolds. In *AISTATS*, 2021.

[6] F. Alvarez, H. Attouch, J. Bolte, and P. Redont. A second-order gradient-like dissipative dynamical system with Hessian-driven damping: Application to optimization and mechanics. *Journal de Mathématiques Pures et Appliquées*, 81(8):747–779, 2002. ISSN 0021-7824. .

[7] H. Attouch and Z. Chbani. Combining fast inertial dynamics for convex optimization with Tikhonov regularization. 2016.

[8] H. Attouch and M. Czarnecki. Asymptotic behavior of gradient-like dynamical systems involving inertia and multiscale aspects. *Journal of Differential Equations*, 262(3):2745–2770, 2017. ISSN 0022-0396. .

[9] H. Attouch, Z. Chbani, J. Fadili, and H. Riahi. First-order optimization algorithms via inertial systems with Hessian driven damping. *Mathematical Programming*, Nov 2020. .

[10] H. Attouch, Z. Chbani, J. M. Fadili, and H. Riahi. Convergence of iterates for first-order optimization algorithms with inertia and Hessian driven damping. *Optimization*, 2021. .

[11] H. Attouch, A. Balhag, Z. Chbani, and H. Riahi. Fast convex optimization via inertial dynamics combining viscous and Hessian-driven damping with time rescaling. *Evolution Equations and Control Theory*, 11(2):487–514, 2022.

[12] P. Attri, Y. Sharma, K. Takach, and F. Shah. Timeseries forecasting for weather prediction. *Keras Tutorial*, 2020. URL https://keras.io/examples/timeseries/timeseries_weather_forecasting/.

[13] A. Beck and M. Teboulle. Gradient-based algorithms with applications to signal-recovery problems. *Convex Optimization in Signal Processing and Communications*, pages 42–88, 2009. .

[14] A. Benettin, G.and Giorgilli. On the Hamiltonian interpolation of near-to-the identity symplectic mappings with application to symplectic integration algorithms. *Journal of Statistical Physics*, 74:1117–1143, 03 1994. .

[15] D. Bertsekas. *Convex Optimization Algorithms*. Athena Scientific, 2009.

[16] M. Betancourt, M. I. Jordan, and A. Wilson. On symplectic optimization. 2018. URL https://arxiv.org/abs/1802.03653.

[17] S. Blanes and F. Casas. *A Concise Introduction to Geometric Numerical Integration*. 2017. ISBN 9781482263442. .

[18] V. Boltyanski, H. Martini, V. Soltan, and V.P. Soltan. *Geometric Methods and Optimization Problems*. Combinatorial Optimization. Springer US, 1999. .

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

[20] J. P. Calvo and J. M. Sanz-Serna. The development of variable-step symplectic integrators, with application to the two-body problem. *SIAM J. Sci. Comp.*, 14(4):936–952, 1993.
[21] C. M. Campos, A. Mahillo, and D. Martín de Diego. A discrete variational derivation of accelerated methods in optimization. 2021. URL https://arxiv.org/abs/2106.02700.
[22] A. L. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Acad. Sci. Paris*, 25:536–538, 1847.
[23] Y.-H. Dai, L.-Z. Liao, and D. Li. On restart procedures for the conjugate gradient method: Theory and practice in optimization. *Numerical Algorithms*, 35, 04 2004. .
[24] A. Deaño, D. Huybrechs, and A. Iserles. *Computing Highly Oscillatory Integrals*. SIAM, Philadelphia, January 2018.
[25] K. Donghwan and J. Fessler. Adaptive restart of the optimized gradient method for convex optimization. *Journal of Optimization Theory and Applications*, 178, 07 2018. .
[26] Z. Drezner and H.W. Hamacher. *Facility Location: Applications and Theory*. Springer Berlin Heidelberg, 2002. ISBN 9783540213451.
[27] T. Duong and N. Atanasov. Hamiltonian-based neural ODE networks on the SE(3) manifold for dynamics learning and control. In *Proceedings of Robotics: Science and Systems*, July 2021. .
[28] V. Duruisseaux and M. Leok. Accelerated optimization on Riemannian manifolds via discrete constrained variational integrators. *Journal of Nonlinear Science*, 32(42), 2022. URL https://doi.org/10.1007/s00332-022-09795-9.
[29] V. Duruisseaux and M. Leok. Time-adaptive Lagrangian variational integrators for accelerated optimization on manifolds,. 2022. URL https://arxiv.org/abs/2201.03774.
[30] V. Duruisseaux and M. Leok. Accelerated optimization on Riemannian manifolds via projected variational integrators. 2022. URL https://arxiv.org/abs/2201.02904.
[31] V. Duruisseaux and M. Leok. A variational formulation of accelerated optimization on Riemannian manifolds. *SIAM Journal on Mathematics of Data Science*, 4(2):649–674, 2022. URL https://doi.org/10.1137/21M1395648.
[32] V. Duruisseaux, J. Schmitt, and M. Leok. Adaptive Hamiltonian variational integrators and applications to symplectic accelerated optimization. *SIAM Journal on Scientific Computing*, 43(4):A2949–A2980, 2021. URL https://doi.org/10.1137/20M1383835.
[33] V. Duruisseaux, T. Duong, M. Leok, and N. Atanasov. Lie group forced variational integrator networks for learning and control of robot systems. 2022. URL https://arxiv.org/pdf/2211.16006.pdf.
[34] W. M. Farr. Variational integrators for almost-integrable systems. *Celestial Mechanics and Dynamical Astronomy*, 102(2):105–118, 2009.
[35] O. Fercoq and Z. Qu. Restarting accelerated gradient methods with a rough strong convexity estimate. Research Report 1609.07358, Télécom ParisTech, 2016. URL https://hal.telecom-paris.fr/hal-02287730.
[36] O. Fercoq and Z. Qu. Adaptive restart of accelerated gradient methods under local quadratic growth condition. *IMA Journal of Numerical Analysis*, March 2019. .
[37] L. N. G. Filon. On a quadrature formula for trigonometric integrals. *Proceedings of the Royal Society of Edinburgh*, 49:38–47, 1930. .
[38] P. Giselsson and S. Boyd. Monotonicity and restart in fast gradient methods. *Proceedings of the IEEE Conference on Decision and Control*, 2015:5058–5063, 02 2015. .
[39] B. Gladman, M. Duncan, and J. Candy. Symplectic integrators for long-time integrations in celestial mechanics. *Celestial Mech. Dynamical Astronomy*, 52:221–240, 1991.
[40] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
[41] E. Hairer. Variable time step integration with symplectic methods. *Applied Numerical Mathematics*, 25(2-3):219–227, 1997.
[42] E. Hairer, C. Lubich, and G. Wanner. Geometric numerical integration illustrated by the Störmer–Verlet method. *Acta Numerica*, 12:399 – 450, 2003.
[43] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2nd edition, 2006.

[44] B. Hall. *Lie Groups, Lie Algebras, and Representations*. Graduate Texts in Mathematics. Springer Cham, second edition, 2015. .

[45] A. Iserles and G. R. W. Quispel. Why geometric numerical integration? In Kurusch Ebrahimi-Fard and María Barbero Liñán, editors, *Discrete Mechanics, Geometric Integration and Lie–Butcher Series*. Springer International Publishing, 2018.

[46] M. Jendoubi and R. May. On an asymptotically autonomous system with Tikhonov type regularizing term. *Archiv der Mathematik*, 95:389–399, 10 2010. .

[47] M. I. Jordan. Dynamical, symplectic and stochastic perspectives on gradient-based optimization. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 523–549. .

[48] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[49] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[50] S. Lall and M. West. Discrete variational Hamiltonian mechanics. *J. Phys. A*, 39(19): 5509–5519, 2006.

[51] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. .

[52] T. Lee, M. Tao, and M. Leok. Variational symplectic accelerated optimization on Lie groups. 2021.

[53] B. Leimkuhler and S. Reich. *Simulating Hamiltonian Dynamics*, volume 14 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2004.

[54] M. Leok and T. Shingel. Prolongation-collocation variational integrators. *IMA J. Numer. Anal.*, 32(3):1194–1216, 2012.

[55] M. Leok and T. Shingel. General techniques for constructing variational integrators. *Front. Math. China*, 7(2):273–303, 2012.

[56] M. Leok and J. Zhang. Discrete Hamiltonian variational integrators. *IMA Journal of Numerical Analysis*, 31(4):1497–1532, 2011.

[57] D. Levin. Procedures for computing one- and two-dimensional integrals of functions with rapid irregular oscillations. *Mathematics of Computation*, 38(158):531–538, 1982.

[58] D. Levin. Fast integration of rapidly oscillatory functions. *Journal of Computational and Applied Mathematics*, 67(1):95–101, 1996. ISSN 0377-0427. .

[59] Y. Liu, F. Shang, J. Cheng, H. Cheng, and L. Jiao. Accelerated first-order methods for geodesically convex optimization on Riemannian manifolds. In *NeurIPS*, volume 30, pages 4868–4877, 2017.

[60] J. E. Marsden and M. West. Discrete mechanics and variational integrators. *Acta Numer.*, 10:357–514, 2001.

[61] M. Muehlebach and M. I. Jordan. A dynamical systems perspective on Nesterov acceleration. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *PMLR*, Long Beach, CA, USA, 2019.

[62] A. S. Nemirovsky and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley - Interscience series in discrete mathematics. Wiley, 1983.

[63] Y. Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[64] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87 of *Applied Optimization*. Kluwer Academic Publishers, Boston, MA, 2004.

[65] Y. Nesterov. Accelerating the cubic regularization of Newton's method on convex problems. *Math. Program.*, 112:159–181, 2008.

[66] B. O'donoghue and E. Candès. Adaptive restart for accelerated gradient schemes. *Found. Comput. Math.*, 15(3):715–732, jun 2015. ISSN 1615-3375. .

[67] A. Orvieto and A. Lucchi. Shadowing properties of optimization algorithms. In *Advances in Neural Information Processing Systems*, volume 32, pages 12692–12703, 2019.

[68] S. Paul and S. Rakshit. Large-scale multi-label text classification. *Keras Tutorial*, 2020.

URL https://keras.io/examples/nlp/multi_label_classification/.
[69] D. L. Phillips. A technique for the numerical solution of certain integral equations of the first kind. *J. ACM*, 9(1):84–97, jan 1962. ISSN 0004-5411. .
[70] M. J. D. Powell. Restart procedures for the conjugate gradient method. *Mathematical Programming*, 12:241–254, 1977.
[71] J. Renegar and B. Grimmer. A simple nearly optimal restart scheme for speeding up first-order methods. *Found. Comput. Math.*, 22(1):211–256, feb 2022. ISSN 1615-3375. .
[72] V. Roulet and A. d'Aspremont. Sharpness, restart, and acceleration. *SIAM Journal on Optimization*, 30(1):262–289, 2020. .
[73] J. A. Sanders, F. Verhulst, and J. Murdock. *Averaging Methods in Nonlinear Dynamical Systems*. Applied Mathematical Sciences. Springer New York, 2007. ISBN 9780387489186.
[74] J. M. Schmitt and M. Leok. Properties of Hamiltonian variational integrators. *IMA Journal of Numerical Analysis*, 38(1):377–398, 03 2017.
[75] J. M. Schmitt, T. Shingel, and M. Leok. Lagrangian and Hamiltonian Taylor variational integrators. *BIT Numerical Mathematics*, 58:457–488, 2018. .
[76] S. Smith, P. Kindermans, C. Ying, and Q. V. Le. Don't decay the learning rate, increase the batch size. 2018.
[77] W. Su, S. Boyd, and E. Candes. A differential equation for modeling Nesterov's Accelerated Gradient method: theory and insights. *Journal of Machine Learning Research*, 17(153): 1–43, 2016.
[78] M. Tao and T. Ohsawa. Variational optimization on Lie groups, with examples of leading (generalized) eigenvalue problems. In *Proceedings of the 23rd International AISTATS Conference*, volume 108 of *PMLR*, 2020.
[79] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246.
[80] A. N. Tikhonov. Solution of incorrectly formulated problems and the regularization method. *Sov. Math., Dokl.*, 5:1035–1038, 1963. ISSN 0197-6788.
[81] A. N. Tikhonov and V. Y. Arsenin. *Solutions of ill-posed problems*. V. H. Winston & Sons, 1977.
[82] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Other Titles in Applied Mathematics. SIAM, 1997. ISBN 9780898719574.
[83] F. Verhulst. *Nonlinear Differential Equations and Dynamic Systems*. 1996. ISBN 978-3-540-60934-6. .
[84] A. Wibisono, A. Wilson, and M. Jordan. A variational perspective on accelerated methods in optimization. *Proceedings of the National Academy of Sciences*, 113(47):E7351–E7358, 2016.
[85] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, 2017.
[86] H. Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150 (5):262–268, 1990. ISSN 0375-9601. .
[87] K. Zare and V. G. Szebehely. Time transformations in the extended phase-space. *Celestial mechanics*, 11:469–482, 1975.
[88] H. Zhang and S. Sra. First-order methods for geodesically convex optimization. In *29th Annual Conference on Learning Theory*, pages 1617–1638, 2016.
[89] H. Zhang and S. Sra. An estimate sequence for geodesically convex optimization. In *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 1703–1723, 2018.
[90] J. Zhang, A. Mokhtari, S. Sra, and A. Jadbabaie. Direct Runge-Kutta discretization achieves acceleration. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
[91] Y. D. Zhong, B. Dey, and A. Chakraborty. Symplectic ODE-Net: learning Hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2019.