Abstract. Geometric numerical integration has recently been exploited to design symplectic accelerated optimization algorithms by simulating the 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.

1. Introduction

The field of symplectic optimization grew out of efforts to generalize Nesterov’s accelerated gradient method [57], which was shown to converge in $O(1/k^2)$ to the minimum of the convex objective function $f$, which is referred to as acceleration, and it improves on the $O(1/k)$ convergence rate exhibited by standard gradient descent methods. This $O(1/k^2)$ convergence rate was shown in [58] to be optimal among first-order methods using only information about $\nabla f$ at consecutive iterates. Nesterov’s algorithm was shown in [67] 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 $O(1/t^2)$ along any trajectory $x(t)$ of this ODE. It was then shown in [74] that in continuous time, an arbitrary convergence rate $O(1/t^p)$ 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 optimization [41], where symplectic discretizations of the Bregman Hamiltonian flow is used to construct accelerated optimization algorithms.

Lagrangian and Hamiltonian flows can also be described variationally, and this and the time-rescaling property of this family were exploited in [30] 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; 29; 53; 78; 79].

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. In Section 6, we show that momentum restarting makes time-adaptivity futile, which allows us to simplify the algorithms. Then, in Sections 7 and 8, we compare the different geometric integrators, and investigate how the computational performance depends on the different parameters, which allows us to reduce the numbers of parameters to tune in practice. In Section 9, we see that temporal looping can avoid instability issues due to numerical precision, and finally in Section 10, we test the resulting algorithms on problems of interest to the machine learning community.

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 $S$ is the functional

$$S(q) = \int_0^T L(q, \dot{q})dt,$$

over the space of smooth curves $q : [0, T] \to \mathcal{Q}$. Hamilton’s variational principle states that $\delta S = 0$ where the variation $\delta S$ is induced by an infinitesimal variation $\delta q$ of the trajectory $q$ that vanishes at the endpoints. Given local coordinates $(q^1, \ldots, q^n)$ on the manifold $\mathcal{Q}$, Hamilton’s variational principle can be shown to be equivalent to the Euler–Lagrange equations,

$$\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}^k} \right) = \frac{\partial L}{\partial q^k}, \quad \text{for } k = 1, \ldots, n. \tag{2.2}$$

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

$$H(q, p) = \left\langle \mathbb{L}(q, \dot{q}), \dot{q} \right\rangle - L(q, \dot{q}) = \sum_{j=1}^n p_j \dot{q}^j - L(q, \dot{q}) \bigg|_{p_i = \frac{\partial L}{\partial \dot{q}^i}}, \tag{2.3}$$

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,

$$\dot{p}_k = -\frac{\partial H}{\partial q^k}(p, q), \quad \dot{q}^k = \frac{\partial H}{\partial p_k}(p, q), \quad \text{for } k = 1, \ldots, n. \tag{2.4}$$

These equations can also be shown to be equivalent to the Euler–Lagrange equations (2.2), provided that 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; 37]. 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 [39] for a brief recent overview of geometric numerical integration, and to [17; 37; 47] for a more comprehensive presentation of structure-preserving integration techniques.
A Type II Hamiltonian variational integrator is constructed by using an approximate discrete variational integrator is defined by constructing an approximation $L(q, p)$ where $q$ denotes a partial derivative with respect to the $i$-th argument. The error analysis is greatly simplified via Theorem 2.3.1 of [54], which states that if a discrete Lagrangian, $L_d : Q \times Q \rightarrow \mathbb{R}$, approximates the exact discrete Lagrangian $L^E_d : Q \times Q \rightarrow \mathbb{R}$ to order $r$, i.e.,

$$L_d(q_0, q_1; h) = L^E_d(q_0, q_1; h) + \mathcal{O}(h^{r+1}),$$

then the discrete Hamiltonian map $F_{L_d} : (q_k, p_k) \mapsto (q_{k+1}, p_{k+1})$ defined by (2.6) and viewed as a one-step method, has order of accuracy $r$. Many other 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 the framework of Type II/III generating functions, referred to as discrete Hamiltonians [44; 50; 64]. 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,

$$H^{+E}_d(q_0, p_1; h) = p_1^Tq_1 - \int_0^h [p(t)^T\dot{q}(t) - H(q(t), p(t))] dt,$$

(2.8)

where $(q, p)$ satisfies Hamilton’s equations with boundary conditions $q(0) = q_0$ and $p(h) = p_1$. A Type II Hamiltonian variational integrator is constructed by using an approximate discrete Hamiltonian $H^+_d$, and applying the discrete right Hamilton’s equations

$$p_0 = D_1 H^+_d(q_0, p_1), \quad q_1 = D_2 H^+_d(q_0, p_1).$$

(2.9)

Theorem 2.3.1 of [54], which simplifies the error analysis for Lagrangian variational integrators, has an analogue for Hamiltonian variational integrators. Theorem 2.2 in [64] states that if a discrete right Hamiltonian $H^+_d$ approximates the exact discrete right Hamiltonian $H^{+E}_d$ to order $r$, i.e.,

$$H^+_d(q_0, p_1; h) = H^{+E}_d(q_0, p_1; h) + \mathcal{O}(h^{r+1}),$$

(2.10)

then the discrete right Hamiltonian map $F_{H^+_d} : (q_k, p_k) \mapsto (q_{k+1}, p_{k+1})$ defined by (2.9) 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 [30; 50]). Examples of variational integrators include Galerkin variational integrators [50; 54], Taylor variational integrators [65], and prolongation-collocation variational integrators [48]. 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 [27; 30].
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 [65]. 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 [64]. 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]. Nesterov’s Accelerated Gradient (NAG) method

\[ x_k = y_{k-1} - h \nabla f(y_{k-1}), \quad y_k = x_k + \frac{k-1}{k+2}(x_k - x_{k-1}), \]

was introduced in 1983 in [57], and converges in \( \Theta(1/k^2) \) to the minimum of the convex objective function \( f \), improving on the \( \Theta(1/k) \) convergence rate exhibited by the standard gradient descent methods. This \( \Theta(1/k^2) \) convergence rate was shown in [58] to be optimal among first-order methods using only information about \( \nabla f \) 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 [56], and accelerated cubic-regularized Newton’s method [59].

It was shown in [67] 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 \( \Theta(1/t^2) \) along the trajectories of this ODE. It was then shown in [74] that in continuous time, the convergence rate of \( f(x(t)) \) can be accelerated to an arbitrary rate \( \Theta(1/t^p) \), 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 \( Q \), given a convex, continuously differentiable function \( h : Q \to \mathbb{R} \) such that \( \| \nabla h(q) \| \to \infty \) as \( \| q \| \to \infty \), its corresponding Bregman divergence is given by

\[ D_h(x, y) = h(y) - h(x) - \langle \nabla h(x), y - x \rangle. \]

The Bregman Lagrangian and Hamiltonian are defined as

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

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

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

\[ \beta_t \leq e^{\alpha t} \quad \text{and} \quad \gamma_t = e^{\alpha t}. \]

If the ideal scaling conditions are satisfied, then Theorem 1.1 in [74] asserts that

\[ f(q(t)) - f(q^*) \leq \Theta(e^{-\beta t}). \]

where \( q^* \) is the desired minimizer of the objective function \( f \).
From now on, we will take \( h(q) = \frac{1}{2} q(q) \). Assuming that the parameter functions \( \alpha, \beta, \gamma \) satisfy the ideal scaling conditions (3.5), the Bregman Lagrangian and Hamiltonian become

\[
L_{\alpha,\beta,\gamma}(q, v, t) = \frac{1}{2} e^{\gamma t - \alpha t} \langle v, v \rangle - e^{\alpha t + \beta t + \gamma t} f(q),
\]

\[
H_{\alpha,\beta,\gamma}(q, r, t) = \frac{1}{2} e^{\alpha t - \gamma t} \langle r, r \rangle + e^{\alpha t + \beta t + \gamma t} f(q),
\]

with corresponding Euler–Lagrange equation given by

\[
\ddot{q}(t) + (e^{\alpha t} - \dot{\alpha}_t) \dot{q}(t) + e^{2\alpha t + \beta t} \nabla f(q(t)) = 0.
\]

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

\[
\alpha_t = \log p - \log t, \quad \beta_t = p \log t + \log C, \quad \gamma_t = p \log t,
\]

where \( C > 0 \) is a constant. These parameter functions satisfy the ideal scaling conditions (3.5), and the corresponding Lagrangian and Hamiltonian are given by

\[
L_p(q, v, t) = \frac{t^{p+1}}{2p} \langle v, v \rangle - C p t^{2p-2} f(q),
\]

\[
H_p(q, r, t) = \frac{p}{2t^{p+1}} \langle r, r \rangle + C p t^{2p-2} f(q),
\]

with corresponding Euler–Lagrange equation given by

\[
\ddot{q}(t) + \frac{p+1}{t} \dot{q}(t) + C p t^{p-2} \nabla f(q(t)) = 0.
\]

From Theorem 1.1 in [74], the evolution \( q(t) \) resulting from this dynamical system satisfies the convergence rate

\[
f(q(t)) - f(q^*) \leq O(1/t^p).
\]

Note that this Bregman subfamily has been exploited extensively in [16; 27; 30; 74], and also note that the special case where \( p = 2 \) and \( C = 1/4 \) corresponds to the limiting continuous differential equation introduced in [67] 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

\[
\alpha_t = \log \eta, \quad \beta_t = \eta t + \log C, \quad \gamma_t = \eta t,
\]

where \( C > 0 \) is a constant. These parameter functions satisfy the ideal scaling conditions (3.5), and the corresponding Lagrangian and Hamiltonian are given by

\[
L_{\eta}(q, v, t) = \frac{e^{\eta t}}{2\eta} \langle v, v \rangle - C \eta e^{2\eta t} f(q),
\]

\[
H_{\eta}(q, r, t) = \frac{\eta}{2 e^{\eta t}} \langle r, r \rangle + C \eta e^{2\eta t} f(q),
\]

with corresponding Euler–Lagrange equation given by

\[
\ddot{q}(t) + C \eta e^{\eta t} \nabla f(q(t)) = 0
\]

From Theorem 1.1 in [74], the evolution \( q(t) \) resulting from this dynamical system satisfies the convergence rate

\[
f(q(t)) - f(q^*) \leq O(e^{-\eta t}).
\]

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** ([74]). *If the curve \( q(t) \) 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 \( L_{\tilde{\alpha},\tilde{\beta},\tilde{\gamma}} \) where*

\[
L_{\tilde{\alpha},\tilde{\beta},\tilde{\gamma}}(q, v, \tau) = \tilde{\tau}(t) L_{\alpha,\beta,\gamma} \left( \frac{1}{\tilde{\tau}(t)} v, \tau(t) \right), \quad \tilde{\alpha}_t = \alpha_{\tau(t)} + \log \tilde{\tau}(t), \quad \tilde{\beta}_t = \beta_{\tau(t)}, \quad \tilde{\gamma}_t = \gamma_{\tau(t)}.
\]

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 [30] with the polynomial subfamily of Section 3.2, where time-rescaling a solution to the \( p \)-Bregman Euler–Lagrange equations via \( \tau(t) = \eta^p \) yielded a solution to the \( \tilde{p} \)-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 \( \tau(t) = \eta^{\frac{1}{n}} t \), or jump from exponential Bregman dynamics to polynomial Bregman dynamics via \( \tau(t) = \frac{\eta}{n} \log t \), and vice-versa via \( \tau(t) = e^{nt/p} \).

However, when symplectic integrators were first used in combination with variable time-steps, they performed poorly [20; 33]. A major advantage of using symplectic integrators on conservative Hamiltonian systems is that they exhibit excellent long-time near-energy preservation [37]. Backward error analysis [37] shows that symplectic integrators can be associated with a 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 [30; 35; 77]. Given a Hamiltonian \( H(q, p, t) \), consider a desired transformation of time \( t \rightarrow \tau \) described by the monitor function \( \frac{d\tau}{dt} = g(t) \). The time \( t \) shall be referred to as the physical time of the system, while \( \tau \) will be referred to as the fictive time. A new Hamiltonian system is constructed using the Poincaré transformation,

\[
\bar{H}(\bar{q}, \bar{p}) = g(q) \left( H(q, q, p) + p \right),
\]

in the extended phase space defined by

\[
\bar{q} = \begin{bmatrix} q \\ \bar{q} \end{bmatrix} \in \bar{Q} \quad \text{and} \quad \bar{p} = \begin{bmatrix} p \\ \bar{p} \end{bmatrix},
\]

where \( \bar{p} \) is the conjugate momentum for \( q = t \) with \( p(0) = -H(q(0), 0, p(0)) \). Then, using a symplectic integrator with constant time-step in fictive time \( \tau \) 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 [30]), 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 [27].
Going back to accelerated optimization, and denoting momentum by \( r \) to avoid confusion, we can jump from one form of Bregman dynamics to another as follows:

1. Polynomial-\( p \) to Polynomial-\( \dot{p} \): \( \tau(t) = \tilde{p}^{\beta/p} \), \( g(t) = \tilde{p}^{1-\beta/p} \), yielding the Poincaré Hamiltonian
   \[
   \dot{H}_{p-\dot{p}}(\bar{q}, \bar{r}) = \frac{P^2}{2p'q+p'/p} (r, r) + \frac{Cp^2}{p'} q^{2p-\beta/p} f(q) + \frac{p}{p'} r q^{1-\beta/p}. \tag{3.23}
   \]

2. Exponential-\( \eta \) to Exponential-\( \dot{\eta} \): \( \tau(t) = \frac{\eta}{\eta} t \), \( g(t) = \frac{\eta}{\eta} \), yielding the Poincaré Hamiltonian
   \[
   \dot{H}_{\eta-\dot{\eta}}(\bar{q}, \bar{r}) = \frac{\eta^2}{2\eta e^{\eta q}} (r, r) + \frac{C\eta^2}{\eta} e^{2\eta q} f(q) + \frac{\eta}{\eta} r. \tag{3.24}
   \]

3. Exponential-\( \eta \) to Polynomial-\( p \): \( \tau(t) = \frac{\eta}{\eta} \log t \), \( g(t) = \frac{\eta}{\eta} t \), yielding the Poincaré Hamiltonian
   \[
   \dot{H}_{\eta-p}(\bar{q}, \bar{r}) = \frac{q^2}{2p e^{\eta q}} (r, r) + \frac{Cp^2}{\eta} q^{2p-1} f(q) + \frac{p}{\eta} r. \tag{3.25}
   \]

4. Polynomial-\( p \) to Exponential-\( \eta \): \( \tau(t) = e^{\eta t/p} \), \( g(t) = \frac{\eta}{\eta} e^{-\eta t/p} \) yielding the Poincaré Hamiltonian
   \[
   \dot{H}_{-\eta-p}(\bar{q}, \bar{r}) = e^{-\eta q} \left( \frac{p^2}{2\eta e^{\eta q}} (r, r) + \frac{Cp^2}{\eta} q^{2p-1} f(q) + \frac{p}{\eta} r \right). \tag{3.26}
   \]

### 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:

\[
L(\bar{q}(\tau), \dot{q}'(\tau)) = q'(\tau) L \left( q(\tau), \frac{q'(\tau)}{g(q(\tau))}, q(\tau) \right) - \lambda(\tau) \left[ q'(\tau) - g(q(\tau)) \right], \tag{3.27}
\]

defined in the extended space \( \bar{q} = (q, q, \lambda)^T \) where time is viewed as a position coordinate \( q = t \), where \( \lambda \) is a Lagrange multiplier enforcing the desired time rescaling \( \frac{dt}{\tau} = g(t) \), and where apostrophes denote derivatives with respect to fictive time \( \tau \). Now, if \( (\bar{q}(\tau), \bar{q}'(\tau)) \) satisfies the Euler–Lagrange equations corresponding to the Lagrangian \( \bar{L} \), then its components satisfy \( \frac{dt}{\tau} = g(t) \) and the original Euler–Lagrange equations \cite{27}.

A discrete variational formulation of these continuous extended Lagrangian mechanics can be formulated \cite{27}, by considering a discrete Lagrangian

\[
L_d(q_k, q_{k+1}, q_{k+1}) \approx \max_{(q, \dot{q}) \in \mathbb{C}^2([\tau_k, \tau_{k+1}], \Omega \times \mathbb{R})} \int_{\tau_k}^{\tau_{k+1}} L \left( q, \frac{q'}{g(q)}, q \right) d\tau. \tag{3.28}
\]

where \( 0 = \tau_0 < \tau_1 < \ldots < \tau_N \) partitions the time interval of interest, and \((q_k, q_{k+1})^{N}_{k=0}\) is a discrete curve in \( \Omega \times \mathbb{R} \) such that \( q_{k+1} = g(q_k) \) and \( q_k = q(\tau_k) \). Defining the discrete momenta via the discrete Legendre transformations, \( p_k = -D_1 L_d(q_k, q_{k+1}, q_{k+1}) \) and using a constant time-step \( h \) fictive time \( \tau \), the corresponding discrete extended Euler–Lagrange equations can be written as

\[
p_k = -D_1 L_d(q_k, q_{k+1}, q_{k+1}),
\]

\[
p_{k+1} = g(q_{k+1}) D_3 L_d(q_k, q_{k+1}, q_{k+1}),
\]

\[
q_{k+1} = q_k + h g(q_k), \tag{3.29}
\]

with two additional equations for \( p_k \) and \( p_{k+1} \) (see \cite{27}). For accelerated optimization, we are usually not interested in the evolution of \( p \), 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
\[ g(t) = \frac{p}{\eta} t^{\frac{1}{\eta} - 1}, \quad g(t) = \frac{\eta}{p}, \quad g(t) = \frac{p t}{\eta}, \quad g(t) = \frac{p e^{-\eta t}}{\eta}, \] (3.30)
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 for designing symplectic integrators for the 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 were 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 sixteen algorithms are presented in Appendix A.

4.1.1. Hamiltonian Taylor Variational Integrator (HTVI). Proceeding as in [30, Section 4.4] or [65], we can derive the Hamiltonian Taylor Variational Integrator (HTVI),
\[ p_{k+1} = p_k - \frac{h}{2} \mathcal{H}_q(q_k, p_{k+1}), \]
\[ q_{k+1} = q_k + \frac{h}{2} \left[ \mathcal{H}_p(q_k, p_{k+1}) + \mathcal{H}_p(q_{k+1}, p_{k+1}) \right], \] (4.1)
These updates recover the Symplectic Euler method [37], which is a popular symplectic integrator of order 1.

4.1.2. Lagrangian Taylor Variational Integrator (LTVI). As in [27], we can define a discrete Lagrangian,
\[ L_d(\bar{q}_0, \bar{q}_1) = \frac{h}{2} \left[ \bar{q}_1 - \bar{q}_0 \right] \frac{\bar{q}_1 - \bar{q}_0}{h g(q_0)}, \] (4.2)
and the updates for the Lagrangian Taylor Variational Integrator (LTVI) can be obtained from the discrete extended Euler–Lagrange equations (3.29).

4.1.3. Störmer–Verlet (SV). A popular symplectic integrator is the Störmer–Verlet (SV) method,
\[ p_{k+1/2} = p_k - \frac{h}{2} \mathcal{H}_q(q_k, p_{k+1/2}), \]
\[ q_{k+1} = q_k + \frac{h}{2} \left[ \mathcal{H}_p(q_k, p_{k+1/2}) + \mathcal{H}_p(q_{k+1}, p_{k+1/2}) \right], \] (4.3)
\[ p_{k+1} = p_{k+1/2} - \frac{h}{2} \mathcal{H}_q(q_{k+1}, p_{k+1/2}), \]
which is a symmetric symplectic integrator of order 2 (see [37]). A very detailed description of the Störmer–Verlet method, its different interpretations, and its beneficial numerical properties can be found in [36]. Note however that in the polynomial and polynomial-to-exponential frameworks, the update for \( q \) in the resulting integrators becomes implicit (see PolySV and PolyToExpSV in Appendix A), 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 idea is to decompose the vector field into its components,

\[
\frac{dq}{d\tau} = \frac{dr}{d\tau}, \quad \frac{dr}{d\tau} = \frac{d\theta}{d\tau}, \quad \frac{d\theta}{d\tau} = \frac{dH}{dq} \frac{dq}{d\tau} + \frac{dH}{dr} \frac{dr}{d\tau} + \frac{dH}{d\theta} \frac{d\theta}{d\tau} = \frac{\partial H}{\partial q} \frac{dq}{d\tau} + \frac{\partial H}{\partial r} \frac{dr}{d\tau} + \frac{\partial H}{\partial \theta} \frac{d\theta}{d\tau} = \mathcal{A} + \mathcal{B} + \mathcal{C} + \mathcal{D},
\]

and then combine the corresponding component dynamics using a symmetric leapfrog composition

\[
\Phi_h = \exp\left(\frac{h}{2} \mathcal{B} \right) \circ \exp\left(\frac{h}{2} \mathcal{C} \right) \circ \exp\left(\frac{h}{2} \mathcal{A} \right) \circ \exp\left(\frac{h}{2} \mathcal{B} \right) \circ \exp\left(\frac{h}{2} \mathcal{A} \right) \circ \exp\left(\frac{h}{2} \mathcal{C} \right) \circ \exp\left(\frac{h}{2} \mathcal{D} \right) \quad (4.4)
\]

which satisfies \(\Phi_h = \exp(hH) + O(h^3)\) (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 [30].

As an example, for

\[
\dot{H}_{p\rightarrow p}(\hat{q}, \hat{r}) = \frac{p^2}{2pq^{p+\hat{p}}}r, \quad H_{p\rightarrow p}(\hat{q}, \hat{r}) = \frac{Cp^2}{p}q^{2p-\hat{p}}f(q) + \frac{p}{p}q^{1-\hat{p}}/
\]

the components of the vector field are given by

\[
\mathcal{A} = \frac{p^2}{pq^{p+\hat{p}}}r, \quad \mathcal{B} = \frac{p}{p}q^{1-\hat{p}}/
\]

\[
\mathcal{C} = -\frac{Cp^2}{p}q^{2p-\hat{p}}f(q), \quad \mathcal{D} = -\frac{\partial H}{\partial q}.
\]

Then, the corresponding component dynamics are given as follows:

- \(\exp(hA)\) yields the update \(q \leftarrow q + h\frac{p^2}{pq^{p+\hat{p}}}r\)
- \(\exp(hC)\) yields the update \(r \leftarrow r - h\frac{Cp^2}{p}q^{2p-\hat{p}}f(q)\)
- \(\exp(hB)\) yields the differential equation \(q' = \frac{p}{p}q^{1-\hat{p}}/\), which can be solved exactly to obtain
  - \(q \leftarrow \left(\frac{q^{1-\hat{p}}}{p} + (\hat{q}^{1-\hat{p}})\right)\)

Note that the updates for \(\exp(hA), \exp(hB),\) and \(\exp(hC)\) do not involve \(t\), and in practice, we are not interested in the evolution of \(t\), so we can simplify the composition into

\[
\Phi_h = \exp\left(\frac{h}{2} \mathcal{C} \right) \circ \exp\left(\frac{h}{2} \mathcal{B} \right) \circ \exp\left(\frac{h}{2} \mathcal{A} \right) \circ \exp\left(\frac{h}{2} \mathcal{B} \right) \circ \exp\left(\frac{h}{2} \mathcal{A} \right) \circ \exp\left(\frac{h}{2} \mathcal{C} \right),
\]

which gives the PolySLC algorithm,

\[
r \leftarrow r - \frac{Cp^2}{2p}h(q^{2p-\hat{p}}f(q),
\]

\[
q \leftarrow \left(\frac{q^{1-\hat{p}}}{p} + (\hat{q}^{1-\hat{p}})\right),
\]

\[
q \leftarrow q + \frac{h\hat{p}^2}{pq^{p+\hat{p}}}r,
\]

\[
q \leftarrow \left(\frac{q^{1-\hat{p}}}{p} + (\hat{q}^{1-\hat{p}})\right),
\]

\[
r \leftarrow r - \frac{Cp^2}{2p}h(q^{2p-\hat{p}}f(q).
\]

We have chosen to place \(\exp(hA)\) in the middle of the symmetric composition \((4.7)\) so that the gradient \(\nabla f\) only needs to be evaluated at the iterates \(\{q_k\}_{k=1}^n\) and can also be used without further computations in stopping criteria or momentum restarting schemes. We have also chosen to place \(\exp(hC)\) at the left-hand and right-hand of the symmetric composition \((4.7)\) so that in practice, we may combine the first and last updates for the vector \(r\) into a single update (instead of only being able to combine the first and last updates for the scalar \(q\) into a single update), and thus save roughly a third of the computational time.
4.1.5. Remarks. It was observed in [30] 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 as well, leveraging higher-order compositions such as Yoshida splittings [76], but it was observed in [30] that these require a larger number of evaluations of the objective function and of its gradient at each step (7 for Yoshida’s fourth-order splitting, and 19 for Yoshida’s sixth-order splitting, for instance). As a consequence, 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 $\text{dom}(f)$ is convex and for any $\lambda \in [0, 1]$,

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y), \quad \forall x, y \in \text{dom}(f),$$

or equivalently if

$$f(y) \geq f(x) + \nabla f(x)^\top(y - x), \quad \forall x, y \in \text{dom}(f).$$

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

$$f(y) \geq f(x) + \nabla f(x)^\top(y - x) + \mu\|y - x\|^2, \quad \forall x, y \in \text{dom}(f).$$

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

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

for various values of the tolerance $\delta$, and solve the following convex problems.

**Problem 1.** Minimize the quartic polynomial

$$f(x) = 1 + \left[(x - 1)^\top \Sigma (x - 1)\right]^2, \quad \text{where} \quad \Sigma_{ij} = 0.9^{|i-j|} \text{ and } x \in \mathbb{R}^d. \quad (4.13)$$

This convex (not strongly convex) function achieves its global minimum at $x^* = (1, 1, \ldots, 1)^\top$.

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

$$f(x_1, x_2) = x_1 + x_2^2 - \ln(x_1x_2), \quad (4.14)$$

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

**Problem 3.** Minimize the strongly convex function

$$f(x_1, \ldots, x_d) = \sum_{k=1}^d x_k \log x_k. \quad (4.15)$$

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

**Problem 4.** Minimize the ill-conditioned strongly convex function

$$f(x_1, x_2, x_3) = 1 + 0.01 x_1^2 + x_2^2 + 100x_3^2, \quad (4.16)$$

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

$$f(x) = \frac{1}{2} x^\top A^\top Ax - b^\top Ax, \quad (4.17)$$
with a gradient given by $\nabla f(x) = A^T Ax - A^T b$. A vector $x \in \mathbb{R}^n$ is a solution of the least squares problem if and only if it satisfies the normal equation $A^T Ax = A^T b$. Furthermore, the least squares problem has a unique solution, given by $x^* = (A^T A)^{-1} A^T b$, if and only if the matrix $A$ has full rank [72].

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 [62; 70; 71] (or $\ell^2$ regularization), where we minimize the convex function

$$f(x) = \|Ax - b\|_2^2 + \lambda \|x\|_2^2,$$  \hspace{1cm} (4.18)

for some $\lambda > 0$, which has a unique minimizer $x^* = (A^T A + \lambda I)^{-1} A^T b$. Another regularized version is the $\ell^1$ penalized linear regression (also known as the Lasso problem [69]), where we minimize the convex (not strongly convex) function

$$f(x) = \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1.$$  \hspace{1cm} (4.19)

**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($w^T x$) is a good model for $y(x)$. This can be formulated as the problem of minimizing the convex (not strongly convex) function

$$f(w) = \sum_{i=1}^{m} \log (1 + \exp(-y_i w^T x_i)).$$  \hspace{1cm} (4.20)

As for linear regression, there are also regularized versions of logistic regression, such as $\ell^1$ and $\ell^2$ regularized logistic regression:

$$f(w) = \sum_{i=1}^{m} \log (1 + \exp(-y_i w^T x_i)) + \lambda \|x\|_1, \quad f(w) = \sum_{i=1}^{m} \log (1 + \exp(-y_i w^T x_i)) + \lambda \|x\|_2^2.$$  \hspace{1cm} (4.21)

**Problem 7 (Fermat–Weber Location Problem [13; 18; 24]).** 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

$$f(x) = \sum_{j=1}^{m} w_j \|x - y_j\|.$$  \hspace{1cm} (4.22)

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

**Remark 4.1.** 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; 40] 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:

$$\ddot{x}(t) + \alpha(t) \dot{x}(t) + \gamma(t) \nabla f(x(t)) + \epsilon(t) x(t) = 0.$$  \hspace{1cm} (4.23)

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

$$L(x, v, t) = \frac{1}{2} \alpha(t) \langle v, v \rangle + \epsilon(t) \langle v, x \rangle - \gamma(t) f(x),$$  \hspace{1cm} (4.24)

$$H(x, p, t) = \frac{1}{2 \alpha(t)} \langle p - \epsilon(t)x, p - \epsilon(t)x \rangle + \gamma(t) f(x),$$  \hspace{1cm} (4.25)

whose Euler–Lagrange equation reads

$$\alpha(t) \ddot{x}(t) + \dot{\alpha}(t) \dot{x}(t) + \gamma(t) \nabla f(x(t)) + \dot{\epsilon}(t) x(t) = 0.$$  \hspace{1cm} (4.26)
5. Controlling the Oscillatory Behavior

The Bregman Euler–Lagrange equation (3.9) can be written in the form

$$\ddot{x}(t) + d(t) \dot{x}(t) + b(t) \nabla f(x(t)) = 0.$$  \hfill (5.1)

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 \cite{55, 67} 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 a momentum restarting scheme. We will consider three different momentum restarting schemes which are inspired from the schemes presented in [60] and [67]:

- **Function Scheme** [60]: 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** [60]: Restart momentum whenever $\nabla f(q_k)(q_k - q_{k-1}) > 0$
  This scheme 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** [67]: 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, as in [67], 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.

![Graphs showing error vs iterations number](image)

**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 $100 \times 100$ 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 $100 \times 100$ logarithmically-spaced grid.

Figure 2 confirms the earlier observations 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. Indeed, we can that 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.

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 families of Bregman dynamics and to the other algorithms, as can be seen, for instance, in Figure 5 for the ExpoLSC algorithm, where momentum restarting leads to significant gains in computational efficiency, robustness and stability.
Figure 4. Contour plot of the number of iterations required to achieve convergence in the \((C, h)\)-plane, for the \(p = \hat{p} = 8\) PolyHTVI algorithm applied to Problem 1.

for Problem 2. Table 1 provides some additional data supporting the significant gain in efficiency that can be achieved using momentum restarting.

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

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 (3.13) and (3.18),

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

(5.2)

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

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

(5.3)

The solutions to these two unperturbed equations are given by

$$u(t) = (k_1 t^{-p} + k_2) 1, \quad \text{and} \quad v(t) = (k_3 e^{-\eta t} + k_4) 1,$$

(5.4)

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 $\Theta(t^{-p})$ and $\Theta(e^{-\eta t})$. 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; 30; 41; 74] 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 [67] 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 [29], where the constant $C$ might not have been optimally tuned in practice (in [26–29; 46; 68] 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 solutions to the Bregman Euler–Lagrange equations 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 for instance 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 the value of $C$ is decreased does not seem to be universal, as can be seen from Figures 9 and 10 for Problem 4.

![Figure 6. Error as a function of time $t$ along the $p = \hat{p} = 6$ polynomial Bregman dynamics for Problem 1, with different values of the constant $C$.](image)

We will now try to obtain a better understanding of the dependence on $C$ of the computational efficiency of the optimization algorithms, and of how a good choice of parameter $C$ depends on the other variables. Preliminary experiments showed that the convergence regions are very similar for the different algorithms within the same adaptive family of Bregman dynamics, so we will only use the HTVI algorithms here but the results extend to the other algorithms. We will return to the comparison of the different geometric integrators later in Section 7.

Let us first investigate whether the regions of optimal convergence in the $(C, h)$-plane are problem-dependent. Figures 11 and 12 display the convergence regions obtained when using the HTVI algorithm for the polynomial and exponential Bregman dynamics on four different objective functions. Unfortunately, these results show that the regions of optimal convergence are problem-dependent and as a result we will not be able to find a single value of $C$ which will achieve almost-optimal performance on all problems.

However, from Figures 13, 14 and 15, we can see that for fixed values of $p, \hat{p}, \eta, \hat{\eta}$, the convergence regions in the $(C, h)$-plane are left almost unchanged as the dimension of the problem is increased.
Figure 7. Error as a function of time $t$ along the $\eta = \bar{\eta} = 0.5$ exponential Bregman dynamics for Problem 1, with different values of the constant $C$.

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

from $d = 3$ to $d = 100$, although the numbers of iterations required increase slightly with $d$. 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.

Note that all the observations made in this section extend to the ExpoToPoly and PolyToExpo subfamilies of time-adaptive Bregman dynamics.
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:

$$\ddot{x}(t) + \gamma(t)\dot{x}(t) + \beta(t)\nabla^2 f(x(t))\dot{x}(t) + b(t)\nabla f(x(t)) = 0.$$  \hspace{1cm} (5.5)

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 $\nabla f$. First-order optimization algorithms were also derived by discretizing the modified differential equation, after rewriting...
\[ \nabla^2 f(x(t)) \dot{x}(t) = \frac{d}{dt} \nabla f(x(t)). \] 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 \cite{38; 49; 50; 54}, 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 $\gamma$ is the decay rate, $\alpha$ 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 \cite{32} or Levin-type \cite{51; 52} quadrature rules in the construction of the integrators, since they are designed specifically for highly oscillatory integrals (see \cite{23} for a thorough presentation).

Another possibility involves averaging techniques \cite{31; 63; 64; 73}. The extended Bregman Hamiltonians or Lagrangians can be split as

$$H(\tilde{q}, \tilde{r}) = H^A(\tilde{q}, \tilde{r}) + CH^B(\tilde{q}),$$  \hspace{1cm} (5.6)

or

$$L(\tilde{q}, \tilde{q}') = L^A(\tilde{q}, \tilde{q}') + CL^B(\tilde{q}),$$  \hspace{1cm} (5.7)

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 $r$. 

---

**Figure 15.** Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-6}$) in the $(C, h)$-plane, for $\eta = 2, \bar{\eta} = 1$ ExpoHTVI applied to Problem 1 with different dimensions $d$. 

---

22 VALENTIN DURUISEAXS AND MELVIN LEOK
6. Time-Adaptivity in the Momentum Restarted Algorithms

We will first investigate how the optimization algorithms behave as the parameters $\hat{p}$ and $\hat{\eta}$ are varied. In [26–30], numerical experiments with the polynomial Bregman dynamics suggested that time-adaptivity (i.e. using $\hat{p} \neq p$) could result in significantly faster optimization algorithms due to the exponentially growing time-steps that they use (instead of constant time-steps for $\hat{p} = p$). These numerical experiments were however carried with the standard versions of the algorithms and without a careful tuning of the parameter $C$.

New numerical experiments carried in this paper suggest that the introduction of momentum restarting schemes in the algorithms enables significantly faster optimization and seems to remove the advantages of the time-adaptive formulation. Indeed, the contour plots in $(C, h)$-space presented in Figures 16, 17, 18, 19, 20, 21 show that the performance and robustness of the PolyHTVI and ExpoHTVI algorithms with momentum restarting is almost unaffected by the introduction of time-adaptivity, regardless of which of Problems 1, 2, 3, 4 they are applied to. This is confirmed by the contour plots in $(\hat{p}, h)$-space and $(\hat{\eta}, h)$-space presented in Figures 22 and 23 where we can see that for fixed $p$ or $\eta$, the value of $\hat{p}$ or $\hat{\eta}$ has very little effect on the performance of the algorithms.

Overall, 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. However, our numerical experiments 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 $\hat{p}$ or $\hat{\eta}$. For this reason, we will now discard time-adaptivity, and focus on the non-adaptive approaches. More precisely, we will not consider the ExpoToPoly and PolyToExpo Bregman subfamilies anymore, and will only focus on the $p = \hat{p}$ polynomial and $\eta = \hat{\eta}$ exponential Bregman subfamilies.

![Figure 16. Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-6}$) in the $(C, h)$-plane, for PolyHTVI applied to Problem 1.](image)

7. Comparison of Integrators

Without time-adaptivity, the accelerated optimization algorithms derived in Section 4.1 can be simplified into the symplectic algorithms presented in Appendix B, and are now all explicit. It
Figure 17. Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-6}$) in the $(C, h)$-plane, for PolyHTVI applied to Problem 2.

Figure 18. Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-6}$) in the $(C, h)$-plane, for PolyHTVI applied to Problem 3.

should be noted that the HTVI and LTVI algorithms are now equivalent, and will be referred to as LTVI from now on (since the construction of LTVIs extends to Riemannian manifolds, while that of HTVIs does not). Note that we are still using momentum restarting based on the gradient scheme in all our numerical experiments.
Figure 19. Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-8}$) in the $(C, h)$-plane, for PolyHTVI applied to Problem 4.

Figure 20. Contour plot of the number of iterations required to achieve convergence ($\delta = 10^{-7}$) in the $(C, h)$-plane, for ExpoHTVI applied to Problem 1.

Regions of convergence in the $(C, h)$ and $(p, h)$ planes for the different algorithms were computed based on 100×100 grids of points and are presented in Figures 24, 25, 26. We can see that the regions of fast convergence both in the $(C, h)$-plane and $(p, h)$-plane for the different algorithms are
almost identical, regardless of the termination criteria. It seems that the three algorithms perform in a very similar way in terms of computational efficiency, robustness, and stability.
We pushed the numerical experimentation further and solved Problems 1, 2, 3, 4 on a 3-dimensional grid of $100^3$ points in $(C, p, h)$-space. The results are displayed in Figures 27, 28, 29, 30.

The top barplots investigate the overall robustness and efficiency of the algorithms. They display the percentages of time each algorithm met the convergence criteria under certain numbers of iterations. For instance, the first bar in the top plot of Figure 27 shows that SV converged in < 50 iterations roughly 5% of the time, in < 100 iterations close to 10% of the time, and so on.

Each bar in the middle barplots compares the performance of two algorithms for a specific problem, by displaying the percentages of times they outperformed each other. For instance, the last bar in the middle plot of Figure 27 shows that SV outperformed LTVI about 21% of the time, while LTVI outperformed SV roughly 11% of the time.

The bottom barplots quantify the gain in efficiency of each algorithm versus the others, by displaying the speedups observed in terms of number of iterations required. For instance, the last bar in the bottom plot of Figure 27 shows that LTVI, when compared to SV, achieves a speedup > 1.5× roughly 4.5% of the time, > 2× roughly 3.5% of the time, etc... Note that for each bar, we only considered triplets $(C, h, p)$ for which both algorithms converged within 10000 iterations.

Let us first focus on the polynomial Bregman family, based on the numerical results displayed in Figures 27 and 28. The top plots confirm the earlier observation that the 3 algorithms have very similar regions of fast convergence, and are thus comparable in terms of robustness. The middle and bottom plots indicate that SLC outperforms SV more often than vice-versa, although both scenarios occur rather rarely. Although LTVI seems to outperform SLC/SV roughly as regularly as vice-versa, the speedups LTVI allows when compared to SLC/SV are not as significant and frequent as the slowdowns it entails. It should also be noted that as the termination criteria are made stricter, the differences and significant speedups between the methods become much rarer (although this is partially due to the fact that smaller tolerances require more iterations and we stopped iterating after 10000 iterations). Overall, the 3 algorithms perform very similarly, but the numerical results presented here suggest that SLC might be slightly better within the polynomial Bregman family.
Let us now focus on the exponential Bregman family, based on the numerical results displayed in Figures 29 and 30. As was the case for the polynomial family, the 3 algorithms perform very similarly in terms of robustness, and the differences in performance between the algorithms become less significant as the convergence criteria are made stricter. SLC and SV perform almost identically on all problems, regardless of the convergence criteria. Now, it seems that LTVI outperforms SLC/SV slightly more often than vice-versa with the more relaxed tolerances, but with less significant speedups, and as the convergence criteria is made stricter, SLC/SV algorithms seem to outperform LTVI. Overall, the 3 algorithms perform very similarly, but the numerical results suggest that SLC/SV might be the slightly better choices for the exponential Bregman family.

To conclude this section, all the algorithms seem to perform very well and with very small discrepancies, but if we had to choose an algorithm to use to integrate the Bregman dynamics, it seems that the SLC algorithms with momentum restarting are the slightly better choice. Algorithms 1, 2 show more detailed pseudocodes for the SLC algorithms:

\begin{algorithm}[H]
\SetAlgoLined
\KwInput{An objective function \(f: \mathbb{R}^d \rightarrow \mathbb{R}\). An initial guess \(q \in \mathbb{R}^d\). Parameters \(C, h, p > 0\).}
\caption{Symmetric Leapfrog Composition of Component Dynamics for the Polynomial Bregman dynamics, with Momentum Restarting (PolySLC-R)}
\begin{algorithmic}[1]
\State \(q \leftarrow 1\), \(G \leftarrow \nabla f(q)\), \(r \leftarrow -\frac{1}{2}Chpq^{2p-1}G\)
\While{convergence criteria are not met}
\State \(\Delta q \leftarrow hp(q + \frac{h}{2})^{p-1}r\)
\State \(q \leftarrow q + \Delta q\)
\State \(G \leftarrow \nabla f(q)\)
\If{\(G^T\Delta q > 0\)} restart momentum: \(r \leftarrow 0\)
\State \(q \leftarrow q + h\)
\State \(r \leftarrow r - Chpq^{2p-1}G\)
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

\begin{algorithm}[H]
\SetAlgoLined
\KwInput{An objective function \(f: \mathbb{R}^d \rightarrow \mathbb{R}\). An initial guess \(q \in \mathbb{R}^d\). Parameters \(C, h, \eta > 0\).}
\caption{Symmetric Leapfrog Composition of Component Dynamics for the Exponential Bregman dynamics, with Momentum Restarting (ExpoSLC-R)}
\begin{algorithmic}[1]
\State \(q \leftarrow 1\), \(G \leftarrow \nabla f(q)\), \(r \leftarrow -\frac{1}{2}C\eta he^{2\eta G}\)
\While{convergence criteria are not met}
\State \(\Delta q \leftarrow \eta he^{-\eta(q + \frac{h}{2})}r\)
\State \(q \leftarrow q + \Delta q\)
\State \(G \leftarrow \nabla f(q)\)
\If{\(G^T\Delta q > 0\)} restart momentum: \(r \leftarrow 0\)
\State \(q \leftarrow q + h\)
\State \(r \leftarrow r - C\eta he^{2\eta G}\)
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

8. Tuning the algorithms

We will now investigate how the PolySLC-R and ExpoSLC-R algorithms perform as the parameters \(C, h, p, \eta\) are varied, and try to reduce the number of parameters needing tuning in practice.

8.1. Tuning PolySLC-R. We solved Problems 1, 2, 3, 4 and two distinct randomly generated instances of Problem 5 using PolySLC-R on a 3-dimensional grid of 500 × 153 × 500 points in \((C, p, h)-\text{space}\) (logarithmically-spaced in \(C\) between \(10^{-12}\) and \(10^{12}\), logarithmically-spaced in \(h\) between \(10^{-6}\) and \(10^{3}\), and linearly-spaced in \(p\) between 2 and 40), and recorded the number of iterations needed to achieve convergence with \(\delta = 10^{-10}\). Figure 31 displays the number of \((C, h)\)
Figure 24. Convergence regions in the $(C, h)$-plane for PolyLTVI, PolySLC and PolySV applied to Problems 1, 2, 3 with $p = 8$. 
Figure 25. Convergence regions in the \((C, h)\)-plane for ExpoLTVI, ExpoSLC and ExpoSV applied to Problems 1, 2 with \(\eta = 6\).

Figure 26. Convergence regions in the \((p, h)\)-plane for PolyLTVI, PolySLC and PolySV applied to Problems 1, 4.
Figure 27. Results for the polynomial Bregman family with $\delta = 10^{-5}$. 
Figure 28. Results for the polynomial Bregman family with $\delta = 10^{-10}$. 
Figure 29. Results for the exponential Bregman family with $\delta = 10^{-5}$. 

PRACTICAL PERSPECTIVES ON SYMPLECTIC ACCELERATED OPTIMIZATION
Figure 30. Results for the exponential Bregman family with $\delta = 10^{-10}$. 
pairs for which convergence was achieved under 200 and 50 iterations for each value of $p$. We can see that the value of $p$ does not seem to significantly affect the number of $(C, h)$ pairs that exhibit fast convergence, once it is taken to be sufficiently large, so tuning the parameter $p$ carefully might not be very helpful and necessary. For numerical stability reasons, which will be discussed in Section 9, it is preferable to use lower values of $p$, so we will set $p = 6$ since this is a small value of $p$ which performed very well in Figure 31.

![Figure 31. Number of $(C, h)$ pairs (out of $500^2$) for which convergence was achieved under 200 and 50 iterations using PolySLC-R, as the value of $p$ is varied.](image)

We solved the same problems using PolySLC-R with $p = 6$ on a 2-dimensional grid of $200 \times 10000$ logarithmically-spaced points in $(C, h)$-space ($C$ between $10^{-15}$ and $10^{15}$, $h$ between $10^{-8}$ and $10^{4}$). The results, presented in Figure 32, confirm the observation made in Section 5.2 that there is no universally optimal value of $C$. However, $C = 0.1$ is an intermediate value which seems to work well for most problems, so we will set it as the default value, but it might need some tuning in practice. A similar experiment was conducted for $10^5$ logarithmically-spaced values of $h$ using PolySLC-R with $p = 6$ and $C = 0.1$, and Figure 33 shows that $h = 0.01$ could be a good default value.

8.2. Tuning ExpoSLC-R. We solved Problems 1, 2, 3, 4 and two distinct randomly generated instances of Problem 5 using ExpoSLC-R on a 3-dimensional grid of logarithmically-spaced $500 \times 100 \times 500$ points in $(C, \eta, h)$-space ($C$ between $10^{-12}$ and $10^{12}$, $h$ between $10^{-6}$ and $10^3$, $\eta$ between $10^{-5}$ and $10^2$), and recorded the number of iterations needed to achieve convergence with $\delta = 10^{-10}$. Figure 34 displays the number of $(C, h)$ pairs for which convergence was achieved under 200 and 50 iterations for each value of $\eta$. As for the polynomial Bregman algorithm, the value of $\eta$ does not seem to significantly affect the number of $(C, h)$ pairs of fast convergence, as long as it falls between 0.001 and 10. Therefore, tuning the parameter $\eta$ carefully might not be very helpful and necessary, so we will fix it to $\eta = 0.01$.

We then solved Problems 1, 2, 3, 4 and two distinct randomly generated instances of Problem 5 using ExpoSLC-R with $\eta = 0.01$ on a 2-dimensional grid of $200 \times 10000$ logarithmically-spaced points
in \((C, h)\)-space \((C\) between \(10^{-15}\) and \(10^{15}\), \(h\) between \(10^{-8}\) and \(10^{4}\)). The results, presented in Figure 35 confirm the observation made in Section 5.2 that there is no value of \(C\) which is universally optimal. However, \(C = 1\) seems to work well for most problems, so we will set it as the default value, but it might need some tuning in practice.

A similar experiment was conducted for \(10^5\) logarithmically-spaced values of \(h\) using PolySLC-R with \(\eta = 0.01\) and \(C = 1\), and Figure 36 shows that \(h = 4\) seems to perform well on most problems considered here.

9. 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 \(q, q\) and \(r\) associated with the exponential Poincaré Hamiltonian,

\[
\bar{H}^\eta(q, r) = \frac{\eta}{2e^{\eta q}}(r, r) + C\eta e^{2\eta q}f(q) + r,
\]

is guided by Hamilton’s equations,

\[
\dot{q} = \eta e^{-\eta q}r, \quad \dot{r} = -C\eta e^{2\eta q}\nabla f(q), \quad \dot{q} = 1.
\]

From these equations of motion, we can see that the time variable \(q\) grows linearly without bound, and as a result quantities like \(e^{\eta q}\) grow exponentially without bound. In practice, this will cause numerical instability due to numerical precision issues, since the unbounded growth of certain terms cannot be balanced by the decay to 0 of other quantities which is limited by machine precision. More precisely, looking at the updates of Algorithm 2,

\[
r \leftarrow r - C\eta he^{2\eta q}\nabla f(q), \quad q \leftarrow q + h, \quad q + \Delta q = q + \eta he^{-\eta(q + \frac{h}{2})}r,
\]

\[
\delta = 10^{-6}
\]

\[
\delta = 10^{-10}
\]
we have at every iteration that

$$ \Delta q \leftarrow A(\Delta q)_{\text{previous}} + Be^{\eta q} \nabla f(q), $$

(9.4)

for some constants $A$ and $B$. The decay to 0 of $\nabla f(q)$ is limited by machine precision, while $e^{\eta q}$ grows without bound, and eventually the quantity $Be^{\eta q} \nabla f(q)$ becomes large again and the position variable $q$ will move 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 term is replaced by an unbounded growing polynomial term.

This numerical instability phenomenon is illustrated in Figure 37 which displays the evolution of the error $|f(x_k) - f(x^*)|$ when the PolySLC-R and ExpoSLC-R algorithms are applied to Problem 2. We can see that both algorithms first achieve convergence to machine precision (after 134 and 154 iterations, respectively), stay at the minimizer for a few hundred iterations, and finally are expelled away from the minimizer due to numerical instability (after 314 and 399 iterations, respectively).

In all our numerical experiments so far, the algorithms stopped when they reached a desired convergence criterion, and as a consequence we did not observe this numerical instability phenomenon as it happens only after convergence is achieved. However, in practice, optimization algorithms are very often terminated after a specified number of iterations instead of a specified tolerance being achieved. Therefore, we need a strategy to avoid this numerical instability phenomenon.
Figure 34. Number of $(C, h)$ pairs (out of 500) for which convergence was achieved under 200 and 50 iterations using ExpoSLC-R, as the value of $\eta$ is varied.

Figure 35. Number of values of $h$ (out of $10^4$) for which convergence was achieved under 200 iterations using ExpoSLC-R with $\eta = 0.01$, as the value of $C$ is varied.
Figure 36. The top two plots display the values of \( h \) for which ExpoSLC-R with \( \eta = 0.01 \) and \( C = 1 \) converged in less than 200 iterations for each of the six problems considered. The bottom two plots display the number of problems (out of 6) that convergence was achieved in less than 200 iterations.

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 \( e^{\eta q} \) grows without bound in the exponential subfamily, it is natural to try to avoid this phenomenon by restricting the growth of the term \( e^{\eta q} \), that is, by restricting the growth of \( q \) (and similarly in the polynomial case).

One possibility is to reset time whenever a certain numerical instability criterion is met, via \( q \leftarrow \beta q \) for some \( \beta \in (0, 1) \). A larger value of \( \beta \) 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 mini-batch approach. It is also preferable to avoid values of \( \beta \) 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 \( \beta \) 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 \( q \leftarrow q - \nu h \) for some \( \nu > 1 \). A smaller \( \nu \) is preferable to retain momentum, while \( \nu \) should not be too close to 1 to avoid numerical instability. In practice, we will reset time via \( q \leftarrow \max(\epsilon, \beta q) \) or \( q \leftarrow \max(\epsilon, q - \nu h) \), where \( \epsilon \) is a small positive number, to avoid very small or negative values.
of time $q$. This phenomenon where the time variable $q$ is stuck in a loop by resetting $q \leftarrow \beta q$ or $q \leftarrow q - \nu h$ whenever numerical instability is near will be referred to as **Temporal Looping**.

Improving the ExpoSLC-R algorithm via temporal looping yields Algorithm 3, which will be referred to as ExpoSLC-RTL:

**Algorithm 3: Symmetric Leapfrog Composition for the Exponential Bregman dynamics, with Momentum Restarting and Temporal Looping (ExpoSLC-RTL)**

**Input:** An objective function $f : \mathbb{R}^d \rightarrow \mathbb{R}$. An initial guess $q \in \mathbb{R}^d$.

Parameters $C, h, p > 0$, $\beta \in (0, 1)$ or $\nu > 0$.

1. $\epsilon \leftarrow 0.001$, $q \leftarrow 1$, $G \leftarrow \nabla f(q)$, $r \leftarrow -\frac{1}{2} C \eta h e^{2\eta q} G$

2. while convergence criteria are not met do
   3. $\Delta q \leftarrow \eta h e^{-\eta q h^2} r$
   4. $q \leftarrow q + \Delta q$
   5. $G \leftarrow \nabla f(q)$
   6. if $G^T \Delta q > 0$ then restart momentum: $r \leftarrow 0$
   7. if numerical instability criterion is met then $q \leftarrow \max(\epsilon, \beta q)$ or $q \leftarrow \max(\epsilon, q - \nu h)$
   8. $q \leftarrow q + h$
   9. $r \leftarrow r - C \eta h e^{2\eta q} G$

In our numerical experiments, we have chosen the following numerical instability criterion to reset time in ExpoSLC-RTL:

$$Ch^2 \eta^2 e^{\eta q} \| G \| > e^{-\eta h} \| \Delta q \|. \quad (9.5)$$

This criterion roughly ensures that $|B|e^{\eta q}\| \nabla f(q) \| < |A|\|(\Delta q)_{\text{previous}}\|$ in equation (9.4), so that the new position update is not significantly larger in norm than the previous position update.

Improving the PolySLC-R algorithm via temporal looping yields the following algorithm:
Algorithm 4: Symmetric Leapfrog Composition for the Polynomial Bregman dynamics, with Momentum Restarting and Temporal Looping (PolySLC-RTL)

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 $\nu > 1$.

1. $\epsilon \leftarrow 0.001$, $q \leftarrow 1$, $G \leftarrow \nabla f(q)$, $r \leftarrow -\frac{1}{2}Chq^2G$
2. while convergence criteria are not met do
3. $\Delta q \leftarrow hp(q + \frac{h}{2})^{-p-1}r$
4. $q \leftarrow q + \Delta q$
5. $G \leftarrow \nabla f(q)$
6. if $G^T\Delta q > 0$ then
7. restart momentum: $r \leftarrow 0$
8. if numerical instability criterion is met then
9. $q \leftarrow \max(\epsilon, \beta q)$ or $q \leftarrow \max(\epsilon, q - \nu h)$
10. $q \leftarrow q + h$
11. $r \leftarrow r - Chq^{2p-1}G$

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

$Ch^2p^2(q + h)^{p+1}\|G\| > q\|\Delta q\|$, \hspace{1cm} (9.6)

which roughly ensures that the new position update is not significantly larger than the previous one. Figure 38 shows that temporal looping takes care of the numerical instability issue:

Figure 38. The effect of temporal looping in PolySLC-R and ExpoSLC-R.

It can be seen from Figures 39 and 40 that temporal looping, with the $q \leftarrow \max(\epsilon, \beta q)$ scheme or $q \leftarrow \max(\epsilon, q - \nu h)$ 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 39](image1.png)

**Figure 39.** 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 40](image2.png)

**Figure 40.** 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 \(\beta\) or \(\nu\), or by designing a better suited numerical instability criterion.
10. Testing for Machine Learning Applications

We now briefly test our algorithms on more challenging optimization problems for machine learning with a variety of model architectures, loss functions to minimize, and applications. For reference, we will also solve these optimization problems using gradient descent and the most commonly used optimizer in machine learning, Adam [42]:

\[
\begin{align*}
    m_{k+1} &= \beta_1 m_k + (1 - \beta_1) \nabla f(x_k) \\
    v_{k+1} &= \beta_2 v_k + (1 - \beta_2) \nabla f(x_k) \odot \nabla f(x_k) \\
    \tilde{m} &= (1 - \beta_1^{k+1})^{-1} m_{k+1}, \quad \tilde{v} = (1 - \beta_2^{k+1})^{-1} v_{k+1} \\
    x_{k+1} &= x_k - h(\sqrt{\tilde{v}} + \epsilon)^{-1} \tilde{m}
\end{align*}
\]

Here, \( u \odot v \) denotes the elementwise multiplication of the vectors \( u \) and \( v \). The variable \( \epsilon \) 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 [42] and PyTorch). The three parameters of Adam are \( \beta_1, \beta_2 \) 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 [42], PyTorch and TensorFlow).

The ExpoSLC-RTL and PolySLC-RTL algorithms have been implemented under the more evocative names eBrAVO and pBrAVO (Bregman Accelerated Variational Optimizer) in Python in such a way that they can be used within the TensorFlow and PyTorch frameworks. These algorithms are available at github.com/vduruss/AccOpt via GNI, and can be called in a similar way as the Adam algorithm in TensorFlow and PyTorch.

In TensorFlow:

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

In PyTorch:

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

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 41 shows the evolution of the loss function (4.20) when trying to formulate a model separating blue and red regions of 2-dimensional space using a line based on the displayed 500 randomly generated points. Figure 42 shows the evolution of the loss function (4.22) when trying to solve 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 41 and 42 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.

Next, we tested our algorithm on the popular multi-label image classification problem based on the Fashion–MNIST dataset [75]: ‘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:
Figure 41. Comparison of algorithms applied to a Binary Classification Problem 6.

Figure 42. Comparison of algorithms applied to a Location Problem 7.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense (Dense)</td>
<td>[-1, 784]</td>
<td>0</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>[-1, 64]</td>
<td>50,240</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>[-1, 64]</td>
<td>0</td>
</tr>
<tr>
<td>dense_3 (Dense)</td>
<td>[-1, 64]</td>
<td>4,160</td>
</tr>
</tbody>
</table>

Total Number of Parameters: 55,050

Figure 43 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 44 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.

We tested our algorithm on another popular multi-label image classification problem based on the CIFAR-10 dataset [43]: 'the CIFAR-10 dataset consists of 60000 32x32 color images in 10 mutually exclusive classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6000 images per class'. We use `nn.CrossEntropyLoss()` as the loss function and a small Convolutional Neural Network in PyTorch very similar to the LeNet-5 architecture described in [45]:
Figure 43. 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 44. 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.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv2d-1</td>
<td>[-1, 6, 28, 28]</td>
<td>456</td>
</tr>
<tr>
<td>Conv2d-2</td>
<td>[-1, 16, 10, 10]</td>
<td>2,416</td>
</tr>
<tr>
<td>Linear-3</td>
<td>[-1, 120]</td>
<td>48,120</td>
</tr>
<tr>
<td>Linear-4</td>
<td>[-1, 84]</td>
<td>10,164</td>
</tr>
<tr>
<td>Linear-5</td>
<td>[-1, 10]</td>
<td>850</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Total Number of Parameters: 62,006

The results are displayed in Figure 45.

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 [61]. 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:
Figure 45. Evolution over 20 epochs of the loss function and accuracy of various algorithms when applied to the CIFAR-10 multi-label image classification problem.

```python
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 sets is displayed in Figure 46. 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 46. 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 as the loss function and the following Long Short-Term Memory (LSTM) model (with 5,153 parameters):

```python
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 47. We can see that the four different algorithms generate similar losses on the training and validation datasets.

![Figure 47. Evolution of the Mean Squared Error on training and validation datasets for several algorithms, when used to optimize a LSTM model for timeseries forecasting for weather prediction.](image)

Then, we solved the following data fitting problem: given 500 data points from a noisy version of the function $10|x|\cos 2x + 10\exp(-\sin x)$ 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:

```python
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'))
model.compile(loss='mse', optimizer=optimizer)
```

The results of this numerical experiment are displayed in Figures 48 and 49. We can see from Figure 48 that all the algorithms achieve very small mean squared error, and this observation is confirmed by Figure 49 which shows that the resulting models, plotted as blue curves, fit the green data points very well.

Finally, we test our algorithms for dynamics learning and control. We consider the Hamiltonian-based neural ODE network from [25] (with 231,310 parameters), inspired by [34; 80], for dynamics learning and control on the rotation group $SO(3)$, applied to a fully-actuated pendulum with dynamics given by $\ddot{\varphi} = -15\sin \varphi + 3u$, where $\varphi$ 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 [80]. We can see from Figure 50 that Adam and the BrAVO algorithms can achieve good training and test losses on this system identification problem, while we were unable to tune SGD to obtain a similar performance.

**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 that was introduced in [74].
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 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 results with accuracy and computational efficiency that are comparable to those of Adam.

Preliminary experiments suggest that the benefits of momentum restarting and temporal looping uncovered in this paper extend to the Riemannian manifold setting introduced in [29]. We intend to explore that direction to improve the computational efficiency and stability of symplectic accelerated optimization algorithms on Riemannian manifolds.

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 [66].

The implementation of the algorithms can also certainly be improved to reduce computational time. One could write different implementations of the algorithm to take advantage of the structure of specific optimization problems or to take advantage of specific computational architectures. 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.

**Data Availability Statement**

Simple implementations of the optimization algorithms in MATLAB and Python, and more sophisticated Python code implementations which allow the optimizers to be called conveniently within the TensorFlow and PyTorch frameworks can be found at

```
github.com/vduruss/AccOpt_via_GNI
```

**Acknowledgments**

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).
PolyHTVI
\[ q_{k+1} = q_k + \frac{p^2}{2p} \eta h \]
\[ r_{k+1} = r_k - \frac{p^2}{2p} \eta h \]
PolySLC
\[ q_{k+1} = q_k + \frac{\eta}{2} h \]
\[ r_{k+1} = r_k - \frac{\eta}{2} Ch_\eta \eta \nabla f(q_k) \]
\[ q_{k+1} = q_k + \frac{\eta}{2} h \]
\[ r_{k+1} = r_k - \frac{\eta}{2} \eta h e^{-\eta \Delta q_k} r_{k+1} \]

ExploHTVI
\[ q_{k+1} = q_k + \frac{\eta}{2} \]
\[ r_{k+1} = r_k - \frac{\eta}{2} Ch_\eta \eta \nabla f(q_k) \]
\[ q_{k+1} = q_k + \frac{\eta}{2} \]
\[ r_{k+1} = r_k - \frac{\eta}{2} \eta h e^{-\eta \Delta q_k} r_{k+1} \]

ExploSV
\[ r_{k+1} = r_k - \frac{\eta}{2} Ch_\eta \eta \nabla f(q_k) \]
\[ q_{k+1} = q_k + \frac{\eta}{2} \]
\[ r_{k+1} = r_k - \frac{\eta}{2} \eta h e^{-\eta \Delta q_k} r_{k+1} \]

PolyLTVI
\[ q_{k+1} = q_k + \frac{p^2}{2p} \eta h \]
\[ r_{k+1} = r_k - \frac{p^2}{2p} \eta h \]

PolySLC
\[ q_{k+1} = q_k + \frac{\eta}{2} h \]
\[ r_{k+1} = r_k - \frac{\eta}{2} \eta h e^{-\eta \Delta q_k} r_{k+1} \]

ExploLTVI
\[ q_{k+1} = q_k + \frac{\eta}{2} \]
\[ r_{k+1} = r_k - \frac{\eta}{2} Ch_\eta \eta \nabla f(q_k) \]
\[ q_{k+1} = q_k + \frac{\eta}{2} \]
\[ r_{k+1} = r_k - \frac{\eta}{2} \eta h e^{-\eta \Delta q_k} r_{k+1} \]

ExploSV
\[ r_{k+1} = r_k - \frac{\eta}{2} Ch_\eta \eta \nabla f(q_k) \]
\[ q_{k+1} = q_k + \frac{\eta}{2} \]
\[ r_{k+1} = r_k - \frac{\eta}{2} \eta h e^{-\eta \Delta q_k} r_{k+1} \]
PolyToExpoHTVI

\[ q_{k+1} = \left(1 + \frac{\eta h}{p}\right)q_k \]
\[ r_{k+1} = r_k - \frac{\eta^2}{p} \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]
\[ q_{k+1} = q_k + \frac{h \eta^2}{p e^{\eta q_k}} q_k \]
\[ r_{k+1} = r_k - \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]

PolyToExpoHTVI

\[ q_{k+1} = \left(1 + \frac{\eta h}{p}\right)q_k \]
\[ r_{k+1} = r_k - \frac{\eta^2}{p} \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]
\[ q_{k+1} = q_k + \frac{h \eta^2}{p e^{\eta q_k}} q_k \]
\[ r_{k+1} = r_k - \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]

PolyToExpoSLC

\[ r_{k+1} = r_k - \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]
\[ q_{k+1} = q_k + \frac{h \eta^2}{p e^{\eta q_k}} q_k \]
\[ r_{k+1} = r_k - \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]

PolyToExpoSLC

\[ r_{k+1} = r_k - \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]
\[ q_{k+1} = q_k + \frac{h \eta^2}{p e^{\eta q_k}} q_k \]
\[ r_{k+1} = r_k - \frac{C h q_k e^{2q_k p} \nabla f(q_k)}{e^{\eta q_k^2} q_k^2} \]
Appendix B. List of Non-Adaptive Algorithms

PolyHTVI

\[ r_{k+1} = r_k - Chpq_k^{2p-1} \nabla f(q_k) \]
\[ q_{k+1} = q_k + hpq_k^{p-1}r_{k+1} \]
\[ q_{k+1} = q_k + h \]

\[ \frac{q_{k+1}}{h} = \frac{q_k}{hp}(q_{k+1} - q_k) \]
\[ q_{k+1} = q_k + h \]

PolySLC

\[ r \leftarrow r - \frac{1}{2} Chpq^{2p-1} \nabla f(q) \]
\[ q \leftarrow q + \frac{h}{2} \]
\[ q \leftarrow q + hpq^{-p}r \]
\[ q \leftarrow q + \frac{h}{2} \]
\[ r \leftarrow r - \frac{1}{2} Chpq^{2p-1} \nabla f(q) \]

PolyLTVI

\[ q_{k+1} = q_k + hpq_k^{-p-1}r_k - Ch^2p^2q_k^{p-2} \nabla f(q_k) \]
\[ r_{k+1} = \frac{q_k}{hp}(q_{k+1} - q_k) \]
\[ q_{k+1} = q_k + h \]

PolySV

\[ r_{k+\frac{1}{2}} = r_k - \frac{1}{2} Chpq_k^{2p-1} \nabla f(q_k) \]
\[ q_{k+1} = q_k + h \]
\[ q_{k+1} = q_k + h \]
\[ r_{k+1} = r_k + \frac{1}{2} Chpq_k^{2p-1} \nabla f(q_{k+1}) \]

ExpoHTVI

\[ r_{k+1} = r_k - C\eta he^{2\eta q_k} \nabla f(q_k) \]
\[ q_{k+1} = q_k + \eta he^{-\eta q_k}r_{k+1} \]
\[ q_{k+1} = q_k + h \]

ExpoSLC

\[ r \leftarrow r - \frac{1}{2} C\eta he^{2\eta q} \nabla f(q) \]
\[ q \leftarrow q + \frac{h}{2} \]
\[ q \leftarrow q + \eta he^{-\eta q}r \]
\[ q \leftarrow q + \frac{h}{2} \]
\[ r \leftarrow r - \frac{1}{2} C\eta he^{2\eta q} \nabla f(q) \]

ExpoLTVI

\[ q_{k+1} = q_k + h\eta e^{-\eta q_k}r_k - C\eta^2h^2e^{\eta q_k} \nabla f(q_k) \]
\[ r_{k+\frac{1}{2}} = e^{\eta q_k} \]
\[ q_{k+1} = q_k + h \]
\[ q_{k+1} = q_k + h \]
\[ r_{k+1} = r_k + \frac{1}{2} C\eta he^{2\eta q_{k+1}} \nabla f(q_{k+1}) \]

ExpoSV

\[ r_{k+\frac{1}{2}} = r_k - \frac{1}{2} C\eta he^{2\eta q_k} \nabla f(q_k) \]
\[ q_{k+1} = q_k + \frac{1}{2} \eta h (e^{-\eta q_{k+1}} + e^{-\eta q_k}) r_{k+\frac{1}{2}} \]
\[ q_{k+1} = q_k + h \]
\[ q_{k+1} = q_k + h \]
\[ r_{k+1} = r_k + \frac{1}{2} C\eta he^{2\eta q_{k+1}} \nabla f(q_{k+1}) \]
References


