2024 American Control Conference (ACC)
July 8-12, 2024. Toronto, Canada

First-Order Dynamic Optimization for Streaming Convex Costs

Mohammadreza Rostami, Hossein Moradian, and Solmaz S. Kia, *Senior member, IEEE*

***Abstract*— This paper proposes a set of novel optimization algorithms for solving a class of convex optimization problems with time-varying streaming cost functions. We develop an approach to track the optimal solution with a bounded error. Unlike prior work, our algorithm is executed only by using the first-order derivatives of the cost function, which makes it computationally efficient for optimization with time-varying cost function. We compare our algorithms to the gradient descent algorithm and show why gradient descent is not an effective solution for optimization problems with time-varying cost. Several examples, including solving a model predictive control problem cast as a convex optimization problem with a streaming time-varying cost function, demonstrate our results.**

time-varying optimization, convex optimization, machine learning, information stream

I. Introduction

In the expansive field of optimization and learning, there is an emerging focus on problems where traditional optimization techniques are not providing efficient solutions on time scales that match the speed of data transmission due to the limitations on computational and communication resources [1]–[10]. Power grids, networked autonomous systems, real-time data processing, learning methods, data-driven control systems, and microelectromechanical systems [11] are among many applications that can stand to benefit from specialized optimization algorithms for optimization problems with time-varying cost. These algorithms are crucial in enabling such applications to recalculate and adjust their optimal operating parameters as shifts in data inputs lead to changes in their cost functions over time. This paper seeks to extend the current understanding and development of optimization algorithms capable of handling time-varying cost functions.

We consider a class of convex optimization programs where the objective function $f : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ is time-varying. More specifically, the optimization problem is given as

$$\mathbf{x}^\star(t) = \arg\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}, t), \quad t \in \mathbb{R}_{\geq 0}. \tag{1}$$

See Fig. 1 for an illustration of the components of (1). Suppose at any $t \in \mathbb{R}_{\geq 0}$, optimization problem (1) is solvable and its minimum value is finite, i.e., $|f(\mathbf{x}^\star(t), t)| = \mathsf{f}_t^\star < \infty$. In what follows, for simplicity, we denote $\mathbf{x}^\star(t)$ by $\mathbf{x}_t^\star$. For optimization problem (1),

$$\nabla_{\mathbf{x}} f(\mathbf{x}_t^\star, t) = \mathbf{0}, \qquad t \in \mathbb{R}_{\geq 0}, \tag{2}$$

is a sufficient condition for $\mathbf{x}_t^\star$ being the optimal trajectory [3].

This work was supported by NSF CAREER award ECCS 1653838. The authors are with the Department of Mechanical and Aerospace Engineering, University of California Irvine, Irvine, CA 92697, {mrostam2,hmoradia,solmaz}@uci.edu.

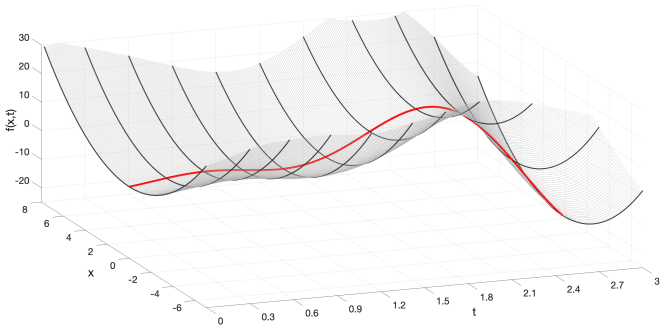


Fig. 1. A time-varying $f(\mathbf{x}, t)$ vs. $\mathbf{x}$ and t (gray plot) and the trajectory of $f(\mathbf{x}^\star(t), t)$ vs. $\mathbf{x}^\star(t)$ and t (red curve).

A trivial way to solve (1) is to sample the cost at particular times, $t_k \in \mathbb{R}_{\geq 0}$, and solve the corresponding sequence of optimization problems assuming $f(\mathbf{x}, t) = f(\mathbf{x}(t_k))$ for each time interval $t \in [t_k, t_{k+1})$. However, this implementation is expected to result in steady-state tracking errors. This tracking error can be significant if the optimal solution is drifting away from $\mathbf{x}_k^\star$ at a rapid pace at the next sampling time.

Recently, [12] and [13] have studied solving the time-varying convex optimization problems from a contraction theory perspective and has provided tracking error bounds between any solution trajectory and the equilibrium trajectory for continuous-time time-varying primal-dual dynamics. In recent work [14], it is shown that for any contracting dynamics dependent on parameters, the tracking error is uniformly upper-bounded in terms of the contraction rate, the Lipschitz constant in which the parameter is involved and the rate of change of the parameter.

On the other hand, for differentiable costs, observing that taking the derivative of (2) results in

$$\dot{\mathbf{x}}_t^\star = -\nabla_{\mathbf{x}\mathbf{x}}^{-1} f(\mathbf{x}_t^\star, t) \nabla_{\mathbf{x}t} f(\mathbf{x}_t^\star, t), \quad t \in \mathbb{R}_{\geq 0}. \tag{3}$$

alternative results, such as those in [2], [3], [15], [16], have proposed continuous-time algorithms aimed at converging to trajectories that satisfy (3) asymptotically, beginning from any initial condition. The literature reviewed so far employs continuous-time dynamics to address the time-varying optimization problem presented in (1). Inspired by the process described in (3), several prediction-correction methods have been introduced in [4], [5], [17], [18] within the discrete-time framework. However, the tracking achieved by these discrete-time algorithms is not exact.

Although elegant, all the aforementioned algorithms suffer from high computational complexity as the algorithms require second-order derivatives of the cost function, as well as

computing the inverse of the Hessian, which is not efficient for high-dimensional problems. Requiring to compute the inverse of Hessian also limits the use of these algorithms for non-convex optimization problems where the inverse may not exist. The use of Hessian has also been proven to be restrictive in the design of distributed optimization algorithms inspired by these second-order algorithms. For example, the algorithms in [3] and [19] require that Hessians of all local objective functions be identical.

In this paper, we present a set of discrete-time optimization algorithms that track the optimal solution of the optimization problem (1) with a quantifiable error bound. Our key contribution lies in designing algorithms that use only the first-order derivatives of the cost function, thereby reducing the computational cost in comparison to the Hessian-based methods. Moreover, use of the first-order derivatives makes our algorithms amenable to solve non-convex time-varying optimization problems. Beside establishing the convergence properties of our algorithms, we compare our algorithms to the gradient descent algorithm and show why gradient descent is not an effective solution for optimization problems with time-varying cost. Due to space constraints, we have omitted the proofs of formal assertions, but interested readers can refer to [20] for these details.

II. Preliminaries

Let the set of real numbers be $\mathbb{R}$. For a vector $\mathbf{x} \in \mathbb{R}^n$ the Euclidean and infinite norms are, respectively, $\|\mathbf{x}\| = \sqrt{\mathbf{x}^\top \mathbf{x}}$ and $\|\mathbf{x}\|_\infty = \max |x_i|_{i=1}^n$. The partial derivatives of a function $f(\mathbf{x}, t) : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ with respect to $\mathbf{x} \in \mathbb{R}^n$ and $t \in \mathbb{R}_{\geq 0}$ are given by $\nabla_{\mathbf{x}} f(\mathbf{x}, t)$ (referred to hereafter as 'gradient') and $\nabla_t f(\mathbf{x}, t)$. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is *m-strongly convex* ($m \in \mathbb{R}_{>0}$) in $\mathbb{R}^d$ if and only if $(\mathbf{z}-\mathbf{x})^\top(\nabla f(\mathbf{z}) - \nabla f(\mathbf{x})) \geq m\|\mathbf{z}-\mathbf{x}\|^2$, for $\forall \mathbf{x}, \mathbf{z} \in \mathbb{R}^d,\ \mathbf{x} \neq \mathbf{z}$. For twice differentiable function f the m-strong convexity is equivalent to $\nabla^2 f(\mathbf{x}) \succeq m\mathbf{I},\ \forall\ \mathbf{x} \in \mathbb{R}^d$. The gradient of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is globally Lipschitz with constant $M \in \mathbb{R}_{>0}$ (hereafter referred to simply as *M-Lipschitz*) if and only if

$$\|\nabla \mathbf{f}(\mathbf{x}) - \nabla \mathbf{f}(\mathbf{y})\| \leq M \|\mathbf{x} - \mathbf{y}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d. \tag{4}$$

For twice differentiable functions, condition (4) is equivalent to $\nabla^2 f(\mathbf{x}) \preceq M\mathbf{I},\ \forall\ \mathbf{x} \in \mathbb{R}^d$ [21]. A differentiable m-strongly function with a globally M-Lipschitz gradient for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ also satisfies [21]

$$\begin{aligned} \nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x}) + \frac{m}{2}\|\mathbf{y}-\mathbf{x}\|^2 \leq f(\mathbf{y}) - f(\mathbf{x}) \\ \leq \nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x}) + \frac{M}{2}\|\mathbf{y}-\mathbf{x}\|^2, \end{aligned} \tag{5a}$$

$$\frac{1}{2M}\|\nabla f(\mathbf{x})\|^2 \leq f(\mathbf{x}) - f^\star \leq \frac{1}{2m}\|\nabla f(\mathbf{x})\|^2, \tag{5b}$$

where $f^\star$ is the minimum value of f.

A. Assumptions on the streaming cost

We pose some well-posedness conditions on the streaming cost.

Assumption 1 (Well-posedness conditions)**.** At any time $t \in \mathbb{R}_{\geq 0}$ and any finite $\mathbf{x} \in \mathbb{R}^n$, we have $|f(\mathbf{x}, t)| < \infty$. Moreover, at any $t \in \mathbb{R}_{\geq 0}$, optimization problem (1) is solvable and its minimum value is finite, i.e., $|f(\mathbf{x}^\star(t), t)| = \mathsf{f}_t^\star < \infty$. □

The optimality condition (2) characterizes the solution of the optimization problem (1). Under the following assumption, the solution $t \mapsto \mathbf{x}_t^\star$ is unique [22].

Assumption 2 (Conditions for the existence of a unique solution)**.** The cost function $f(\mathbf{x}, t) : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ is twice continuously differentiable with respect to $\mathbf{x}$ and continuously differentiable with respect to t and globally Lipschitz in t. Moreover, the cost function is m-strongly convex and has M-Lipschitz gradient in $\mathbf{x}$, i.e.,

$$m\,\mathbf{I}_n \preceq \nabla_{\mathbf{xx}} f(\mathbf{x}, t) \preceq M\,\mathbf{I}_n, \quad t \in \mathbb{R}_{\geq 0}. \qquad \square$$

Beside Assumption 2, to establish our convergence, we place some other conditions on the cost as stated in Assumption 3 below. Note that similar assumptions are also used in the existing second-order time-varying optimization algorithms.

Assumption 3 (Smoothness of the cost function)**.** The function $f(\mathbf{x}, t) : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ is continuously differentiable and sufficiently smooth. Specifically, there exists a bound on the derivative of $f(\mathbf{x}, t)$ as

$$\|\nabla_{\mathbf{x}t} f(\mathbf{x}, t)\| \leq K_2, \quad \nabla_t f(\mathbf{x}, t)| \leq K_1, \quad |\nabla_{tt} f(\mathbf{x}, t)| \leq K_3$$

for any $\mathbf{x} \in \mathbb{R}^n$, $t \in \mathbb{R}_{\geq 0}$. □

By virtue of [23, Lemma 3.3], Assumption 3 leads to cost function $f(\mathbf{x}, t)$ and its first derivatives $\nabla_t f(\mathbf{x}, t)$ and $\nabla_{\mathbf{x}} f(\mathbf{x}, t)$ being globally Lipschitz in $t \in \mathbb{R}_{\geq 0}$ with respective constants K_1, K_2 and K_3.

In what follows, we let $f^\star(t) = f(\mathbf{x}^\star(t), t)$, be the optimal cost value at any $t \in \mathbb{R}_{\geq 0}$.

Lemma II.1 (Bound on the cost difference of the optimizer)**.** Consider the optimization problem (1) under Assumptions 2 and 3. Then,

$$|f^\star(t_{k+1}) - f^\star(t_k)| \leq \psi, \tag{6}$$

where $\psi = \delta(K_1 + \frac{\delta}{2}K_3) + \frac{K_2^2\delta^2}{2m}(\frac{M\delta}{m} + 2)$, with $\delta = t_{k+1} - t_k \in \mathbb{R}_{>0}$ being the sampling timestep of the optimal cost function across time. □

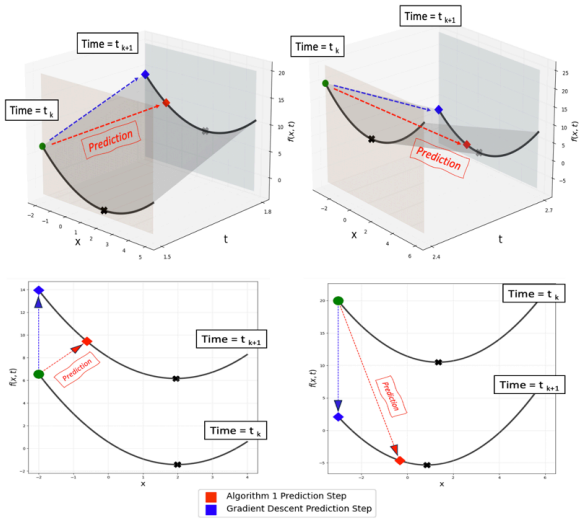
III. Objective statement

For a first-order solver for problem (1), some work such as [24] investigated use of the conventional gradient descent algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\, \nabla_{\mathbf{x}} f(\mathbf{x}_k, t_{k+1}). \tag{7}$$

Considering the first-order approximation of the cost at t_{k+1},

$$f(\mathbf{x}_{k+1}, t_{k+1}) \approx f(\mathbf{x}_k, t_{k+1}) + \nabla_{\mathbf{x}} f(\mathbf{x}_k, t_{k+1})^\top (\mathbf{x}_{k+1} - \mathbf{x}_k), \tag{8}$$



(a) $\nabla_t f(\mathbf{x}_k, t_k) \geq 0$ (b) $\nabla_t f(\mathbf{x}_k, t_k) < 0$

Fig. 2. An example case that demonstrates the role of the prediction step (line 3) of Algorithm 1. As we can see in this example, for both cases of $\nabla_t f(\mathbf{x}_k, t_k) \geq 0$ (plots in the left column) and $\nabla_t f(\mathbf{x}_k, t_k) < 0$ (plots in the right column) the statement of Lemma IV.1 holds, i.e., $f_1^-(t_{k+1}) \leq f_{\mathrm{g}}^-(t_{k+1})$.

the gradient decent algorithm (7) certainly results in function reduction at each t_{k+1}. On the other hand, considering the first-order approximation across time from t_k to t_{k+1},

$$f(\mathbf{x}_k, t_{k+1}) \approx f(\mathbf{x}_k, t_k) + \nabla_t f(\mathbf{x}_k, t_k)^\top \delta. \tag{9}$$

where $\delta = t_{k+1} - t_k$, we see that the gradient descent algorithm (7) is oblivious to $\nabla_t f(\mathbf{x}_k, t_k)^\top (t_{k+1} - t_k)$ and how cost is changing across time. Iterative optimization algorithms for unconstrained problems are driven by successive descent objective. However, if $\nabla_t f(\mathbf{x}_k, t_k)^\top > 0$, we have $f(\mathbf{x}_k, t_{k+1}) > f(\mathbf{x}_k, t_k)$ and thus the function reduction at t_{k+1} after taking the gradient descent algorithm (7) is not the same as if $f(\mathbf{x}_k, t_{k+1}) \approx f(\mathbf{x}_k, t_k)$ (in first-order sense). Thus, one can anticipate that the gradient descent algorithm will result in poor tracking performance during periods of time that $\nabla_t f(\mathbf{x}(t), t) > 0$.

Let us re-write the gradient decent algorithm (7) as

$$\mathbf{x}_{k+1}^- = \mathbf{x}_k, \tag{10a}$$
$$\mathbf{x}_{k+1} = \mathbf{x}_{k+1}^- - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}_{k+1}^-, t_{k+1}). \tag{10b}$$

We refer to $\mathbf{x}_{k+1}^-$ as *predicted* decision variable at time t_{k+1} and to $\mathbf{x}_{k+1}$ as the *updated* decision variable at t_{k+1}. We expect that at each time t_{k+1}, the updated decision variable gets close to $\mathbf{x}^\star(t_{k+1})$ by virtue of function descent. In the subsequent sections, we set to design alternative algorithms which consider the variation of the cost across time and employ prediction rules that will result in a better tracking performance than that of the gradient descent algorithm.

IV. First-order algorithms

In this section, we propose two classes of first-order algorithms to solve optimization problem (1), using prediction

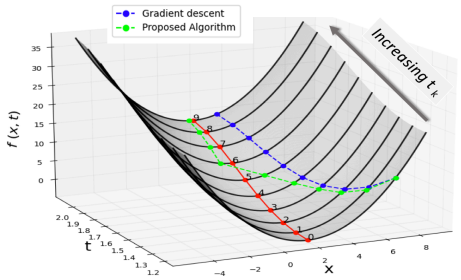


Fig. 3. The difference between the trajectories of the proposed Algorithm 1 and the gradient descent algorithm shown in time interval $t \in [1.2, 2.1]$ for the time-varying cost shown in Fig. 1.

and update steps that take advantage of the first-order term $\nabla_t f(\mathbf{x}_k, t_k)$ to improve convergence performance. Algorithm 1 is our first proposed algorithm. The structure of this algorithm consists of two steps: the *prediction step* changes the local state based on the rate of change of the cost function with respect to time. The subsequent *update step* is a gradient descent step at freeze time t_{k+1}. The following Lemma reveals the advantage of the prediction step of Algorithm 1 over the gradient descent algorithm (10a), which lacks prediction oversight.

Lemma IV.1. Consider the gradient descent algorithm (10a) and Algorithm 1. Let $\delta = (t_{k+1} - t_k)$ be the same for both algorithms. Let $f_{\mathrm{g}}^-(t_{k+1})$ and $f_1^-(t_{k+1})$ be, respectively, the function value of the gradient descent algorithm and Algorithm 1 after the prediction step. Suppose $\mathbf{x}_k$ for both algorithms is the same. Then, for any $\epsilon \in \mathbb{R}_{\geq 0}$, we have $f_1^-(t_{k+1}) \leq f_{\mathrm{g}}^-(t_{k+1})$ in first-order approximate sense.

Proof: The first-order approximation of $f(\mathbf{x}_{k+1}^-, t_{k+1})$ is $f(\mathbf{x}_{k+1}^-, t_{k+1}) \approx f(\mathbf{x}_k, t_k) + \nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)(\mathbf{x}_{k+1}^- - \mathbf{x}_k) + \nabla_t f(\mathbf{x}_k, t_k)(t_{k+1} - t_k)$. For gradient descent algorithm by substitution we obtain $f_{\mathrm{g}}^-(t_{k+1}) \approx f(\mathbf{x}_k, t_k) + \delta \nabla_t f(\mathbf{x}_k, t_k)$. For Algorithm 1, for $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)\| < \epsilon$, we have $f_{\mathrm{g}}^-(t_{k+1}) = f_1^-(t_{k+1}) \approx f(\mathbf{x}_k, t_k) + \delta \nabla_t f(\mathbf{x}_k, t_k)$. For $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)\| \geq \epsilon$, on the other hand we have $f_1^-(t_{k+1}) \approx f(\mathbf{x}_k, t_k) + \nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)(\mathbf{x}_{k+1}^- - \mathbf{x}_k) + \delta \nabla_t f(\mathbf{x}_k, t_k) = f(\mathbf{x}_k, t_k) + \delta (\nabla_t f(\mathbf{x}_k, t_k) - |\nabla_t f(\mathbf{x}_k, t_k)|)$, confirming $f_1^-(t_{k+1}) \leq f_{\mathrm{g}}^-(t_{k+1})$, and completing the proof.

See Fig. 3 for trajectories generated by the gradient descent algorithm and Algorithm 1, which shows Algorithm 1 results in a lower tracking error. Next, we present the convergence analysis of Algorithm 1.

Theorem IV.1 (Convergence analysis of Algorithm 1)**.** Let Assumptions 2 and 3 hold. Then, Algorithm 1 converges to the neighborhood of the optimum solution of (1) with the following upper bound

$$\begin{aligned} f(\mathbf{x}_{k+1}, t_{k+1}) - f^\star(t_{k+1}) \leq & \frac{\left(1 - (1 - 2\kappa\alpha m)^k\right)}{4\kappa^2\alpha^2 m}\psi \\ & + \frac{\left(1 - (1 - 2\kappa\alpha m)^k\right)}{4\kappa^2\alpha^2 m^2} \max(\gamma\delta^2, \mu\delta) \\ & + (1 - 2\kappa\alpha m)^k (f(\mathbf{x}_0, t_0) - f^\star(t_0)), \end{aligned} \tag{11}$$

Algorithm 1 $\epsilon-$exact Time-Varying Optimization with $\nabla_t f(\mathbf{x}_k, t_k)$

Initialization $\mathbf{x}_0 \in \mathbb{R}^n$, $\epsilon, \delta \in \mathbb{R}_{>0}, f(\mathbf{x}_0, t_0) \in \mathbb{R}$,
if $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)\| \geq \epsilon$ **then**
  $\mathbf{x}_{k+1}^- = \mathbf{x}_k - \delta \frac{|\nabla_t f(\mathbf{x}_k, t_k)|}{\|\nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)\|^2} \nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)$
else
  Set $\mathbf{x}_{k+1}^- = \mathbf{x}_k$
end if
Update $f(\mathbf{x}_{k+1}^-, t_{k+1})$
$\mathbf{x}_{k+1} = \mathbf{x}_{k+1}^- - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}_{k+1}^-, t_{k+1})$.

Algorithm 2 $\epsilon-$exact Time-Varying Optimization with $\nabla_t f(\mathbf{x}_k, t_k)$ Approximation

Initialization $\mathbf{x}_0 \in \mathbb{R}^n$, $\epsilon \in \mathbb{R}_{>0}, f(\mathbf{x}_0, t_0) \in \mathbb{R}$,
if $\|\nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)\| \geq \epsilon$ **then**
  $\mathbf{x}_{k+1}^- = \mathbf{x}_k - \frac{|f(\mathbf{x}_k, t_k) - f(\mathbf{x}_k, t_{k-1})|}{\|\nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)\|^2} \nabla_{\mathbf{x}} f(\mathbf{x}_k, t_k)$
else
  Set $\mathbf{x}_{k+1}^- = \mathbf{x}_k$
end if
Update $f(\mathbf{x}_{k+1}^-, t_{k+1})$
$\mathbf{x}_{k+1} = \mathbf{x}_{k+1}^- - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}_{k+1}^-, t_{k+1})$.

where $\psi = \delta(K_1 + \frac{\delta}{2}K_3) + \frac{K_2^2\delta^2}{2m}(\frac{M\delta}{m} + 2)$, $\delta = t_{k+1} - t_k$, $\gamma = \frac{2K_1}{\delta} + \frac{M}{2\epsilon^2}K_1^2 + \frac{1}{2}K_3 + \frac{1}{\epsilon}K_1K_2$, $\mu = K_1 + \frac{\delta}{2}K_3$ and $\kappa = (1 - \alpha M/2)$, provided that $0 < \alpha \leq \frac{1}{2M}$. □

For proof see [20].

Remark IV.1 (Ultimate tracking bound of Algorithm 1)**.** The tracking bound of Algorithm 1 is given by (11). As $k \to \infty$ we have $(1 - 2\kappa\alpha m)^k \to 0$ in (11). Thus, the effect of initialization error $f(\mathbf{x}_0, t_0) - f^\star(t_0)$ vanishing with time. Moreover, the ultimate bound on $f(\mathbf{x}_{k+1}, t_{k+1}) - f^\star(t_{k+1})$ as $k \to \infty$ is $\frac{\psi}{4\kappa^2\alpha^2 m} + \frac{\max(\gamma\delta^2, \mu\delta)}{4\kappa^2\alpha^2 m^2}$. Thus the optimal value of ϵ corresponding to the lowest bound in (11) is obtained as the solution of $\gamma\delta^2 = \mu\delta$, which can be calculated numerically.

Algorithm 1 requires explicit knowledge of $\nabla_t f(\mathbf{x}_k, t_k)$ which may not be available for costs constructed from streaming data. For such problems we propose Algorithm 2, which follows the same prediction-correction structure of Algorithm 2 but uses an approximation for $\nabla_t f(\mathbf{x}_k, t_k)$. In Algorithm 2, we approximate $\nabla_t f(\mathbf{x}_k, t_k)$ by

$$\nabla_t f(\mathbf{x}_k, t_k) \approx \frac{f(\mathbf{x}_k, t_k) - f(\mathbf{x}_k, t_{k-1})}{t_k - t_{k-1}}. \tag{12}$$

Higher-order differences can also be used to construct a better approximation of $f(\mathbf{x}_k, t_k)$, but at the expense of higher computation and storage costs. In both Algorithm 1 and Algorithm 2, $\epsilon \in \mathbb{R}_{>0}$ is an arbitrary chosen parameter that, as we show below, can be tuned to achieve a desired level of tracking accuracy. Next, theorem explains the convergence guarantee of Algorithm 2.

Theorem IV.2 (Convergence analysis of the Algorithm 2)**.** Let Assumptions 2 and 3 hold. Then, the Algorithm 2 converges to the neighborhood of the optimum solution of 1 with the following upper bound

$$\begin{aligned} f(\mathbf{x}_{k+1}, t_{k+1}) - f^\star(t_{k+1}) \leq & \frac{\left(1 - (1 - 2\kappa\alpha m)^k\right)}{4\kappa^2\alpha^2 m}\psi \\ & + \frac{\left(1 - (1 - 2\kappa\alpha m)^k\right)}{4\kappa^2\alpha^2 m^2}\max(\gamma'\delta^2, \mu\delta) \\ & + (1 - 2\kappa\alpha m)^k (f(\mathbf{x}_0, t_0) - f^\star(t_0)), \end{aligned} \tag{13}$$

where $\psi = \delta(K_1 + \frac{\delta}{2}K_3) + \frac{K_2^2\delta^2}{2m}(\frac{M\delta}{m} + 2)$, $\gamma' = K_3 + \frac{2K_1}{\delta} + \frac{K_1^2 M}{\epsilon^2} + \frac{K_2(K_1 + \frac{\delta}{2}K_3)}{\epsilon} + \frac{\delta^2 K_3^2 M}{4\epsilon^2}$, $\mu = K_1 + \frac{\delta}{2}K_3$ and $\kappa = (1 - \alpha M/2)$, provided that $0 < \alpha \leq \frac{1}{2M}$. □

For proof see [20]. Based on Theorem IV.2, a similar statement to Remark IV.1 can be made about the ultimate tracking bound of Algorithm 2. Notice also that the tracking error of Algorithm 2, as one can expect based one Algorithm 2's use of an estimate for $\nabla_t f(\mathbf{x}_k, t_k)$, can be larger than Algorithm 1's because $\gamma' \geq \gamma$.

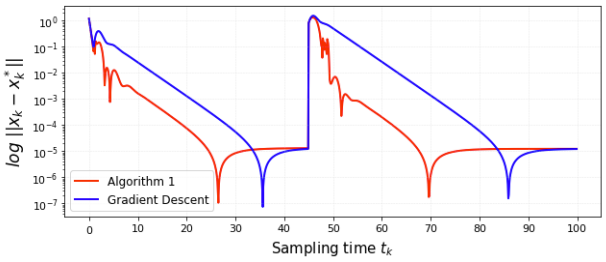


Fig. 4. Log error of the performance of Algorithm 1 versus gradient descent algorithm with respect to the sampling time t_k.

V. Numerical Example

In this section, we demonstrate the performance of Algorithm 1 and Algorithm 2 using three different examples. In the first example we assume that the time derivative of cost is explicitly available. In the next two examples, which include solving a Model Predictive Control (MPC) design as a time-varying convex optimization problem and a learning problem with streaming data, the explicit knowledge about the time derivative of the cost is not available. Therefore, we solve these problems using Algorithm 2.

A. A case of time-varying cost function whose time derivatives are available explicitly

Consider the time-varying cost function $f(\mathbf{x}, t) = (x_1 + x_2 - 0.01)^2 + (1 + \mathrm{e}^{-(t-\tau)})x_2^2 + \mathrm{e}^{-(t-\tau)}x_1 \cdot \sin(2t)$, where $\tau = \left\{ \begin{array}{ll} 0, & \text{if } t < 45 \\ 45, & \text{if } t \geq 45 \end{array} \right\}$. Here, the purpose of the variable τ is to produce a jump within the function at the time instant $t = 45$ to observer the response of the algorithm in relation to abrupt changes in the underlying problem. Although, we can find $t \to \mathbf{x}_t^\star$ explicitly in this problem, we use this problem as a demonstration case.

As depicted in Fig. 4, initializing both gradient descent and Algorithm 1 from $\mathbf{x}_0 = [0.1, 1.2]^\top$, selecting $\alpha = 0.04$,

$\delta = 0.1$ and $\epsilon = 0.03$, for $t < 45$, Algorithm 1 demonstrates the ability to attain an error level of 10^{-3} at $t_k = 10.7$, whereas the gradient descent algorithm achieves the same error threshold at $t_k = 24.7$. In other words, the gradient descent algorithm requires 140 more iterations to reach the error level of 10^{-3}. Furthermore, for $t \geq 45$, after inducing a jump within the function at $t = 45$, Algorithm 1 demonstrates the ability to reach an error level of 10^{-3} at $t_k = 54$, whereas the gradient descent algorithm achieves the same error threshold at $t_k = 71$. In other words, the gradient descent algorithm requires 170 more iterations to reach the error level of 10^{-3}. This simulation demonstrates that Algorithm 1 has a higher convergence rate than the gradient descent algorithm outside the ϵ error bound. However, after both algorithms are in the ϵ error bound and steady-state tracking is achieved, as expected, Algorithm 1 behaves similar to the gradient descent algorithm.

B. Model Predictive Control

MPC consists of solving repeated optimization problems over some fixed moving horizon. These repeated optimization problems can be viewed as an incidence of an optimization problem with streaming/time-varying cost. Let us demonstrate by designing an MPC controller for a unicycle robot, where we want a point of interest h, $[x_h \;\; y_h]^\top = [x + b\cos(\theta), \;\; y + b\sin(\theta)]^\top$, on the robot that a camera is mounted on. (x, y, θ) is the pose vector of the robot. The camera should follow a desired trajectory $r(t) = [r_x(t), r_y(t)]^\top$. This desired trajectory is not available a priori and at each time t it is constructed/adjusted from observing the path using the robots camera system for some forward horizon. The dynamics governing the motion of point h is given by $\begin{bmatrix} \dot{x}_h \\ \dot{y}_h \end{bmatrix} = \begin{bmatrix} \dot{x} - b\dot{\theta}\sin(\theta) \\ \dot{y} + b\dot{\theta}\cos(\theta) \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$. Discretizing this dynamics we obtain $\begin{bmatrix} x_h(k+1) \\ y_h(k+1) \end{bmatrix} = \begin{bmatrix} x_h(k) + \delta u_1 \\ y_h(k) + \delta u_2 \end{bmatrix}$. Then, an MPC-based tracking controller can be obtained from solving the following optimization problem at each time step k, executing only controller $u(k)$ and repeating the process:

$$\min_{\mathbf{u}_1 \in \mathbb{R}^{10}} J_1(k) = \sum_{i=0}^{H_p + H_w - 1} (r_x(k+i) - x_h(k+i))^2 + \frac{1}{\lambda} \sum_{i=0}^{H_u - 1} (u_1(k+i))^2, \tag{14a}$$

$$\min_{\mathbf{u}_2 \in \mathbb{R}^{10}} J_2(k) = \sum i = 0^{H_p + H_w - 1} (r_y(k+i) - y_h(k+i))^2 + \frac{1}{\lambda} \sum_{i=0}^{H_u - 1} (u_2(k+i))^2. \tag{14b}$$

Here, H_p and H_u are the length of the prediction horizon and the control horizon, respectively. Also, H_w may be used to alter the control horizon but for the case of simplicity, we put $H_w = 0$. Here, λ is the weight factor which is assumed to be constant over the prediction horizon. For our numerical example we use the following values $\lambda = 0.1$, $H_p = H_u = 10$. Additionally, we initialize the robot from $x(0) = -100$ and $y(0) = -100$ and setting $u_1(i) = u_2(i) = 1, \; \forall i \in \{0, 1, ..., 9\}$ when $k = 0$.

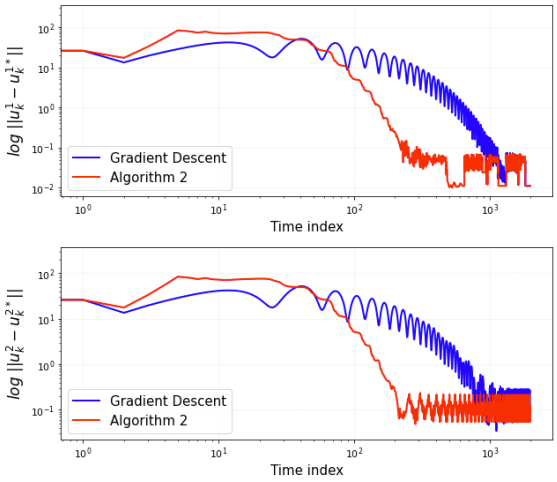


Fig. 5. Log error of the controller u_1 and u_2 with respect to time under executing Algorithm 2 and gradient descent algorithm. Here $u_k^{l\star}$, $l \in 1, 2$ is the optimal control value which given that the costs are quadratic can be obtained analytically by assuming enough computational power.

One of the main challenges with the MPC control is the cost of solving the optimization problems of the form (14) repeatedly with, for example, the gradient descent algorithm which converges asymptotically at each time step. Viewing the optimization problem (14) as a representation of a time-dependent cost at the specific time t_k, we address the MPC issue using Algorithm 2, with the values of $\alpha = 0.01$, $\epsilon = 0.1$, and $\delta = 0.1$ configured for the sampling interval.Note here that since we do not have explicit knowledge of the time derivative of the cost, we cannot solve this problem with Algorithm 1. The results are shown in Fig. 5 which shows that Algorithm 2 achieves a better tracking error than gradient descent algorithm. Meaning that we can take less steps to get to a certain tracking error threshold than the gradient descent algorithm, resulting in reducing computation cost.

C. Learning for streaming data

As a final demonstration example, we analyze the performance of Algorithm 2 in comparison to the gradient descent and other first-order static algorithms for a learning problem. This numerical example is modeled after the first numerical example of [5]. This is an example in which the exact value of $\nabla_t f(\mathbf{x}_k, t_k)$ is not available due to the online arrival of the data and the lack of advanced knowledge about the cost. Suppose that data points arrive sequentially at intervals of $\tau > 0$ where τ can be selected as the inter-arrival time of data. The goal is to find the optimal solution of the regression problem at time $t \in T$ based on data $\mathbf{Z}(t) = \{z(\tau), \tau \in W_t\}$ where W_t is a sliding window. The problem can be formulated as

$$\mathbf{x}^\star(t) = \arg\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}, \mathbf{Z}(t)), \quad t \in \mathbb{R}_{>0},$$

where f is the quadratic least square cost. Here, we consider an example of a 50 dimensional time-varying least-

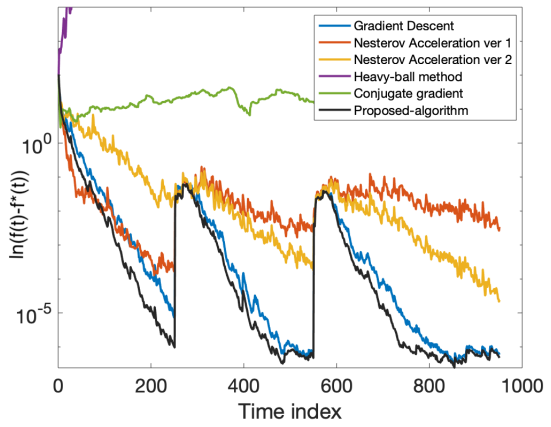


Fig. 6. The performance of different algorithms on tracking the optimal objective over an example of a 50 dimensional time-varying least-squares problem, defined using a sliding window of 50 data points, for 950 time points. Two big jumps in the solution near time indices 250 and 550 are by design. Nesterov ver. 1 does not use knowledge of strong convexity, while ver. 2 does. The non-linear conjugate gradient exploits the quadratic objective to have an exact line-search.

squares problem, defined using a sliding window of 50 data points, for 950 time points. By design, two large jumps in the solution near time indices 250 and 550 are generated. Figure 6 shows that our proposed Algorithm 2 with $\epsilon = 0.01$ outperforms all the known first-order algorithms for static optimization that were simulated in [5], even the accelerated algorithms. This can be attributed to the use of implicit knowledge of $\nabla_t f$ in our algorithm, which gives it an anticipatory mechanism about the changes of the cost with time. Figure 6 is in the semi-logarithmic scale over time. Interestingly, the gradient descent algorithm converges faster than the well-known accelerated algorithms for the time-varying optimization problem. In addition, the proposed algorithm, as shown, outperforms all common optimization algorithms in terms of convergence.

VI. Conclusion

In this paper, we proposed two algorithms for a class of convex optimization problems where the cost is time-varying. Our solutions can track the optimal trajectory of the minimizer with bounded steady-state error. Our algorithms are executed only by using the cost function's first-order derivatives, making them computationally efficient for optimization with a time-varying cost function and amenable to non-convex optimization problems. We argued why problems with time-varying cost should not be solved by gradient descent and how taking into account how the cost function varies with time can lead to first-order optimization algorithms with lower tracking error. We demonstrated the effectiveness of our proposed algorithms through several examples including an MPC problem and a learning task with streaming data. Future work will investigate the distributed implementation of our proposed algorithms for in-network problems where agents communicate according to a graph topology.

References

[1] Q. Ling and A. Ribeiro, "Decentralized dynamic optimization through the alternating direction method of multipliers," *IEEE Transactions on Signal Processing*, vol. 62, no. 5, pp. 1185–1197, 2013.

[2] M. Fazlyab, S. Paternain, and A. Ribeiro, "Prediction-correction interior-point method for time-varying convex optimization," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1973–1986, 2017.

[3] S. Rahili and W. Ren, "Distributed continuous-time convex optimization with time-varying cost functions," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 1590–1605, 2017.

[4] A. Simonetto, A. Mokhtari, A. Koppel, G. Leus, and A. Ribeiro, "A class of prediction-correction methods for time-varying convex optimization," *IEEE Transactions on Signal Processing*, vol. 64, no. 17, pp. 4576–4591, 2016.

[5] E. Dall'Anese, A. Simonetto, S. Becker, and L. Madden, "Optimization and learning with information streams: Time-varying algorithms and applications," *IEEE Signal Processing Magazine*, vol. 37, pp. 71–83, 2020.

[6] F. Malandrino, G. D. Giacomo, A. Karamzade, M. Levorato, and C. F. Chiasserini, "Matching dnn compression and cooperative training with resources and data availability," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, pp. 1–10, 2023.

[7] A. Esteki and S. S. Kia, "Distributed optimal resource allocation with time-varying quadratic cost functions and resources over switching agents," in *European Control Conference*, pp. 441–446, 2022.

[8] M. Rostami and S. S. Kia, "Federated learning using variance reduced stochastic gradient for probabilistically activated agents," in *2023 American Control Conference (ACC)*, IEEE, 2023.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," PMLR, 2017.

[10] Y. Ruan, X. Zhang, S. C. Liang, and C. Joe-Wong, "Towards flexible device participation in federated learning," PMLR, 2021.

[11] D. Dadkhah and S. R. Moheimani, "Combining h and resonant control to enable high-bandwidth measurements with a mems force sensor," *Mechatronics*, vol. 96, p. 103086, 2023.

[12] P. Cisneros-Velarde, S. Jafarpour, and F. Bullo, "A contraction analysis of primal-dual dynamics in distributed and time-varying implementations," *IEEE Transactions on Automatic Control*, 2022.

[13] D. H. Nguyen, L. V. Thanh, T. Konstantin, and S. Jean-Jacques, "Contraction and robustness of continuous time primal-dual dynamics," *IEEE Control Systems Letters*, 2018.

[14] A. Davydov, V. Centorrino, A. Gokhale, G. Russo, and F. Bullo, "Contracting dynamics for time-varying convex optimization," *arXiv preprint arXiv:2305.15595*, 2023.

[15] E. C. Hall and R. M. Willett, "Online convex optimization in dynamic environments," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 647–662, 2015.

[16] Y. Ding, J. Lavaei, and M. Arcak, "Escaping spurious local minimum trajectories in online time-varying nonconvex optimization," *American Control Conference*, pp. 454–461, 2021.

[17] N. Bastianello, A. Simonetto, and R. Carli, "Primal and dual prediction-correction methods for time-varying convex optimization," *arXiv:2004.11709 Available: http://arxiv.org/abs/2004.11709*, 2020.

[18] N. Bastianello, "tvopt: A python framework for time-varying optimization," *IEEE Int. Conf. on Decision and Control*, 2021.

[19] B. Huang, Y. Zou, Z. Meng, and W. Ren, "Distributed time-varying convex optimization for a class of nonlinear multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 801–808, 2020.

[20] M. Rostami, H. Moradian, and S. Kia, "First-order dynamic optimization for streaming convex costs," *arXiv preprint arXiv:2310.07925*, 2023.

[21] Y. Nesterov, *Lectures on convex optimization*, vol. 137. Springer, 2018.

[22] A. L. Dontchev and R. T. Rockafellar, *Implicit functions and solution mappings*. 2009.

[23] H. K. Khalil, *Nonlinear Systems*. Englewood Cliffs, NJ: Prentice Hall, 3 ed., 2002.

[24] A. Y. Popkov, "Gradient methods for nonstationary unconstrained optimization problems," *Automation and Remote Control*, vol. 66, no. 6, pp. 883–891, 2005.