COMMUN. MATH. SCI.
Vol. 16, No. 7, pp. 1933–1973

A SPLITTING METHOD FOR OVERCOMING THE CURSE OF DIMENSIONALITY IN HAMILTON-JACOBI EQUATIONS ARISING FROM NONLINEAR OPTIMAL CONTROL AND DIFFERENTIAL GAMES WITH APPLICATIONS TO TRAJECTORY GENERATION[*]

ALEX TONG LIN[†], YAT TIN CHOW[‡], AND STANLEY J. OSHER[§]

Abstract. Recent observations have bridged splitting methods arising from optimization, to the Hopf and Lax formulas for Hamilton-Jacobi Equations. This has produced extremely fast algorithms in computing solutions of these PDEs. More recent observations were made in generalizing the Hopf and Lax formulas to state-and-time-dependent cases. In this article, we apply a new splitting method based on the Primal Dual Hybrid Gradient algorithm (a.k.a. Chambolle-Pock) to nonlinear optimal control and differential games problems, based on techniques from the derivation of the new Hopf and Lax formulas, which allow us to compute solutions at specified points directly, i.e. without the use of grids in space. This algorithm also allows us to create trajectories directly. Thus we are able to lift the curse of dimensionality a bit, and therefore compute solutions in much higher dimensions than before. And in our numerical experiments, we actually observe that our computations scale polynomially in time. Furthermore, this new algorithm is embarrassingly parallelizable.

Keywords. Hamilton-Jacobi equations; optimal control; differential games; optimization; trajectory generation; Hopf-Lax formula; nonconvex Hamiltonian; nonsmooth Hamiltonian; convex-concave games.

AMS subject classifications. 35F21; 49K20; 49K35; 49L99; 49M99; 49N70; 65M25; 90C26; 90C46; 90C47; 93B40.

1. Introduction

Hamilton-Jacobi equations (HJE) are crucial in solving and analyzing problems arising from optimal control, differential games, dynamical systems, calculus of variations, quantum mechanics, and the list goes on [18, 33].

Most methods to compute HJE use grids and finite-difference discretization. Some of these methods use ENO/WENO-type methods [35], and others use Dijkstra-type methods [11] such as fast marching [44] and fast sweeping [43]. But due to their use of grids, they suffer from the curse of dimensionality, i.e. they do not scale well with increases in dimension in the space variable, i.e. they generally scale exponentially.

In past years, there has been an effort to mitigate the effects of dimensionality on computations of HJE. Some recent attempts to solve Hamilton-Jacobi equations use methods from low rank tensor representations [25], or methods based on alternating least squares [42], or methods by sparse grids [28], or methods using pseudospectral [41] and iterative methods [27]. There have also been attempts to mitigate the curse of dimensionality which have been motivated by reachability [1, 31]. In this work, we examine and advertize the effectiveness of splitting to solve Hamilton-Jacobi equations and to directly compute optimal trajectories.

We note that splitting for optimal control problems was used by [32] (2013), where they applied it to cost functionals with a quadratic and convex term. In terms of

[*]Received: March 10, 2018; Accepted (in revised form): July 2, 2018. Communicated by Jack Xin. This work was partially supported by DOE-SC0013838, NGA HM0210410003, and ECCS-1462398.

[†]Department of Mathematics, University of California, Los Angeles, USA (atlin@math.ucla.edu). www.math.ucla.edu/ãtlin

[‡]Department of Mathematics, University of California, Riverside, USA (ytchow@math.ucla.edu). www.math.ucla.edu/ỹtchow

[§]Department of Mathematics, University of California, Los Angeles, USA (sjo@math.ucla.edu). www.math.ucla.edu/s̃jo

Hamilton-Jacobi equations, Kirchner et al. [30] (2018) have effectively applied PDHG [14, 48] (a.k.a. Chambolle-Pock [3]) to Hamilton-Jacobi equations arising from linear optimal control problems. They applied splitting to the Hopf formula to compute HJE for bounded input, high-dimensional linear control problems. Another main feature of their methods is that they are able to directly generate optimal trajectories by making use of the the closed-form solution to linear ODEs. See also previous work by Kirchner et al. [29] where they apply the Hopf formula to differential games problems, which resulted in complex "teaming" behavior even under linearized pursuit-evasion models.

In this current paper, we have worked in parallel with the above authors and have also applied splitting to Hamilton-Jacobi equations arising from nonlinear problems. Our volunteered method has some nice properties: (1) relatively quick computations of solutions in high dimensions (see Section 6.4, although one can easily extend to 100 dimensions for example, and also see Section 6.1.3 where we observe a linear relationship between computation time and dimension), especially when we include parallelization, the method is embarrassingly parallelizable [23], (2) the ability to *directly generate* optimal trajectories of the optimal control/differential games problems, (3) the ability to compute problems with nonlinear ODE dynamics, (4) the ability to compute solutions for *nonconvex and nonsmooth* Hamiltonians and initial conditions, (5) the ease of parallelization of our algorithm to compute solutions to HJE, i.e. each core can use the algorithm to compute the solution at a point, so given N cores we can compute solutions of the HJE at N points simultaneously, and (6) the ease of parallelization to directly compute trajectories, i.e. in our discretization of the time, we can parallelize by assigning each computational core a point in the time discretization.

Our work lies in using the techniques used to derive the generalized Hopf and Lax formulas introduced by Y.T. Chow et al. [8], which generalize to the state-and-time-dependent cases (note in the literature that the classical Lax formula is sometimes called the Hopf-Lax formula). See also previous work from the same authors, [6, 7], and also [9, 10] where they provide fast algorithms under convexity assumptions. To perform the optimization, we use a new splitting method that is based on the Primal Dual Hybrid Gradient (PDHG) method (a.k.a. Chambolle-Pock), which we *conjecture* to both converge to a local minimum for most well-behaved problems, and which we *conjecture* to also approximate the solution. To do this, we discretize the optimal control problem and the differential games problem in time, a technique inspired by [32] and [8]. This new splitting method has been experimentally seen (Section 6) to be faster than the using coordinate-descent in most cases, which the authors in [8] use to compute the solutions.

As far as the authors know, the use of splitting as applied to minimax differential games problems, mainly on the state-and-time dependent Equations (4.6) and (4.7), is new. In this case, we seem to be able to compute HJE with nonconvex Hamiltonians and nonconvex initial data (Section 6.3.3), although they do have the structure of being convex-concave.

The paper is organized as follows:

- Section 2 gives brief overviews of Hamilton-Jacobi equations and its intimate connections to optimal control (Section 2.1) and differential games (Section 2.2).
- Section 3 gives a brief overview of splitting methods from optimization, focusing on ADMM (Section 3.1) and PDHG (Section 3.2).
- Section 4 presents the generalized Lax and Hopf formulas for optimal control and differential games that were conjectured by [8]. We also go through its discretization in Section 4.1, which is the basis of our algorithm.

- Section 5 **presents the main algorithms.**
- Section 6 presents various computational examples.
- Section 7 ends with a brief conclusion, and a discussion on future work.
- The appendix gives a more in-depth explanation on how to use the algorithms.

2. Hamilton-Jacobi equations and its connection to optimal control and differential games

2.1. Hamilton-Jacobi equations and optimal control.

Most of our exposition on optimal control will follow [17], and also [16] (Chapter 10).

The goal of optimal control theory is to find a control policy that will drive a system while optimizing a criterion. Given an initial point $x \in \mathbb{R}^n$ and an initial time $t \in [0,T]$, where T is some fixed end-point time, the system will obey an ODE

$$\begin{cases} \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s),\boldsymbol{\alpha}(s),s), \quad t < s < T \\ \mathbf{x}(t) = x \end{cases}$$

where $\mathbf{f}: (\mathbb{R}^n \times A \times \mathbb{R}) \to \mathbb{R}$, where $A \subseteq \mathbb{R}^m$. We call $\mathbf{x}$ the *state*, and $\boldsymbol{\alpha}$ the *control.* And the functional we want to optimize is $J_{x,t}: \mathcal{A} \to \mathbb{R}$ where

$$J_{x,t}[\boldsymbol{\alpha}] := g(\mathbf{x}(T)) + \int_t^T L(\mathbf{x}(s),\boldsymbol{\alpha}(s),s)\,ds. \tag{2.1}$$

and where $\mathcal{A} := \{\boldsymbol{\alpha}: [t,T] \to A\}$ is some *admissible control* set, and $g: \mathbb{R}^n \to \mathbb{R}$ and $L: (\mathbb{R}^n \times A \times \mathbb{R}) \to \mathbb{R}$. We can either *minimize* the above functional, in which we call it a *cost*, or we can *maximize* it, in which we call it a *payoff.* For our exposition, we will choose to minimize $J_{x,t}[\cdot]$, so it will be a cost. Then we define the *value function*

$$\varphi(x,t) = \min_{\alpha(\cdot)\in\mathcal{A}} J_{x,t}[\boldsymbol{\alpha}]. \tag{2.2}$$

Under some mild conditions on $\mathbf{f}$, g, and L, this value function will satisfy the *terminal-valued* Hamilton-Jacobi PDE (HJ PDE)

$$\begin{cases} \partial_t\varphi(x,t) + H(x,\nabla_x\varphi(x,t),t) = 0, \qquad (x,t) \in \mathbb{R}^n \times (0,T) \\ \qquad\qquad\qquad\qquad \varphi(x,T) = g(x). \end{cases}$$

where $H(x,p,t) = \min_{a\in A}\{\langle \mathbf{f}(x,a,t),p\rangle + L(x,a,t)\}$.

To get an initial-valued PDE, we can make a change of variables $t \to T - t$. Or equivalently we can reformulate the optimal control problem "backwards in time" so that we have

$$\begin{cases} \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s),\boldsymbol{\alpha}(s),s), \quad 0 < s < t \\ \mathbf{x}(t) = x \end{cases}$$

and

$$J_{x,t}[\boldsymbol{\alpha}] := g(\mathbf{x}(0)) + \int_0^t L(\mathbf{x}(s),\boldsymbol{\alpha}(s),s)\,ds.$$

Then our $\varphi(x,t) = \min_{\alpha(\cdot)\in\mathcal{A}} J_{x,t}[\boldsymbol{\alpha}]$ will satisfy an *initial-valued* HJ PDE with $H(x,p,t) = \max_{a\in A}\{\langle \mathbf{f}(x,a,t),p\rangle - L(x,a,t)\}$. Note that if $\mathbf{f} = a$, then this form of the Hamiltonian expresses H as the convex conjugate [36] of L.

If we think from a physical perspective in which time moves forward, the first formulation feels more comfortable. If we comes from the fields of PDE or mathematical optimization, the latter formulation will feel more comfortable.

So how does having a HJ PDE help us synthesize an optimal control? Using the first formulation as it feels more physically intuitive, we can heuristically argue that given an inital time $t \in (0,T]$ and a state $x \in \mathbb{R}^n$, we consider the optimal ODE

$$\begin{cases} \dot{\mathbf{x}}^*(s) = \mathbf{f}(\mathbf{x}^*(s), \boldsymbol{\alpha}^*(s), s), & t < s < T \\ \mathbf{x}^*(t) = x \end{cases}$$

where at each time $s \in (0,T)$, we choose the value of $\alpha^*(s)$ to be such that

$$\begin{aligned} &\langle \mathbf{f}(\mathbf{x}^*(s), \boldsymbol{\alpha}^*(s), s), \partial_x \varphi(\mathbf{x}^*(s), s)\rangle + L(\mathbf{x}^*(s), \boldsymbol{\alpha}^*(s), s) \\ =& \min_{a \in A} \{ \langle \mathbf{f}(\mathbf{x}^*(s), a, s), \partial_x \varphi(\mathbf{x}^*(s))\rangle + L)(\mathbf{x}^*(s), a, s)\} \\ =& H(\mathbf{x}^*(s), \partial_x \varphi(\mathbf{x}^*(s), s), s). \end{aligned}$$

We call $\boldsymbol{\alpha}^*(\cdot)$ defined in this way as the *feedback control*, and this can be obtained from $\nabla_x \varphi$ (see Section 10.3.3 of [16]). This is also related to Pontryagin's maximum principle (Chapter 4 of [17]).

Note that for the case $H(x,p,t) = H(p)$, we have available the (classical) Hopf and Lax formulas which are expressions for the solutions $\varphi(x,t)$ of the HJ PDE:

When the Hamiltonian $H(p)$ is convex and the initial data g is (uniformly) Lipschitz continuous, then we have the Lax formula:

$$\varphi(x,t) = \min_{y \in \mathbb{R}^n} \left\{ g(y) + t H^* \left(\frac{x-y}{t} \right) \right\}$$

where $H^*(x) = \max_{v \in A} \{\langle v, x\rangle - H(v)\}$ is the convex conjugate of L.

And if the initial data g is (uniformly) Lipschitz continuous and convex, and H is continuous, then we have the Hopf formula,

$$\varphi(x,t) = \sup_{y \in \mathbb{R}^n} \{-g^*(y) + \langle y, x\rangle - tH(y)\} = (g^*(y) + tH(y))^* \tag{2.3}$$

where g^* is the convex conjugate of g. Note the last equality implies the solution is convex in x. We note again that the argument minimum of the above expression is in fact $\nabla_x \varphi(x,t)$.

2.2. Hamilton-Jacobi equations and differential games. Our exposition of differential games will follow [17] (Chapter 6), but also see [26, 47]. In the field of differential games, we restrict our exposition to *two-person, zero-sum differential games*. Let an initial point $x \in \mathbb{R}^n$ and an initial time $t \in [0,T]$ be given, where T is some fixed endpoint time. A two-person, zero-sum differential game will have the dynamics,

$$\begin{cases} \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s), & t < s < T \\ \mathbf{x}(t) = x \end{cases}$$

where $\mathbf{f}\colon (\mathbb{R}^n \times A \times B \times \mathbb{R}) \to \mathbb{R}$, and where $A \subseteq \mathbb{R}^m$ and $B \subseteq \mathbb{R}^\ell$. The control $\boldsymbol{\alpha}$ is the control for player I, and the control $\boldsymbol{\beta}$ is the control for player II. The functional will be $J_{x,t} : A(t) \times B(t) \to \mathbb{R}$ where

$$J_{x,t}[\boldsymbol{\alpha}, \boldsymbol{\beta}] := g(\mathbf{x}(T)) + \int_t^T L(\mathbf{x}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s)\, ds. \tag{2.4}$$

and where $A(t) := \{\boldsymbol{\alpha} : [t,T] \to A\}$ and $B(t) := \{\boldsymbol{\beta} : [t,T] \to B\}$ are *admissible control* sets, and $g : \mathbb{R}^n \to \mathbb{R}$ and $L : (\mathbb{R}^n \times A \times B \times \mathbb{R}) \to \mathbb{R}$.

In order to model that at each time, neither player has knowledge of the other's future moves, we use a concept of strategy that was used by Varaiya [45] as well as Elliot and Kalton [13]. This idea allows us to model that each player will select a control in response to all possible controls the opponent can select.

A *strategy* for player I is a mapping $\Phi : B(t) \to A(t)$ such that for all times $s \in [t,T]$

$$\tau \in [t,s], \quad \boldsymbol{\beta}(\tau) \equiv \widehat{\boldsymbol{\beta}}(\tau) \qquad \text{implies} \qquad \Phi[\boldsymbol{\beta}](\tau) \equiv \Phi[\widehat{\boldsymbol{\beta}}](\tau).$$

The $\Phi[\boldsymbol{\beta}]$ models player I's response to player II selecting $\boldsymbol{\beta}$. We similarly define a strategy $\Psi : A(t) \to B(t)$ for player II:

$$\tau \in [t,s], \quad \boldsymbol{\alpha}(\tau) \equiv \widehat{\boldsymbol{\alpha}}(\tau) \qquad \text{implies} \qquad \Psi[\boldsymbol{\alpha}](\tau) \equiv \Psi[\widehat{\boldsymbol{\alpha}}](\tau)$$

and $\Psi[\boldsymbol{\alpha}]$ models player II's response to player I selecting $\boldsymbol{\alpha}$.

Letting $\mathcal{A}(t)$ and $\mathcal{B}(t)$ be the set of strategies for player I and player II, respectively, then we define the *lower value function* as

$$\varphi^-(x,t) = \inf_{\Psi[\cdot] \in \mathcal{B}(t)} \sup_{\boldsymbol{\alpha}(\cdot) \in A(t)} J_{x,t}[\boldsymbol{\alpha}, \Psi[\boldsymbol{\alpha}]] \tag{2.5}$$

and the *upper value function* as

$$\varphi^+(x,t) = \sup_{\Phi[\cdot] \in \mathcal{A}(t)} \inf_{\boldsymbol{\beta}(\cdot) \in B(t)} J_{x,t}[\Phi[\boldsymbol{\beta}], \boldsymbol{\beta}]. \tag{2.6}$$

Note that we always have $\varphi^-(x,t) \le \varphi^+(x,t)$ for all $x \in \mathbb{R}^n$ and $t \in [0,T]$. For the proof, see [18].

These value functions satisfy the *terminal-valued* HJ PDEs

$$\begin{cases} \partial_t \varphi^-(x,t) + \max_{a \in A} \min_{b \in B} \{\langle \mathbf{f}(x,a,b,t), \nabla_x \varphi^-(x,t)\rangle + L(x,a,b,t)\} = 0 \\ \varphi^-(x,T) = g(x) \end{cases}$$

and

$$\begin{cases} \partial_t \varphi^+(x,t) + \min_{b \in B} \max_{a \in A} \{\langle \mathbf{f}(x,a,b,t), \nabla_x \varphi^+(x,t)\rangle + L(x,a,b,t)\} = 0 \\ \varphi^+(x,T) = g(x) \end{cases}$$

where we have the *lower PDE Hamiltonian*

$$H^-(x,p,t) = \max_{a \in A} \min_{b \in B} \{\langle \mathbf{f}(x,a,b,t), p\rangle + L(x,a,b,t)\}$$

and the *upper PDE Hamiltonian*

$$H^+(x,p,t) = \min_{b \in B} \max_{a \in A} \{\langle \mathbf{f}(x,a,b,t), p\rangle + L(x,a,b,t)\}$$

In general, we have

$$\max_{a \in A} \min_{b \in B} \{\langle \mathbf{f}(x,a,b,t), p\rangle + L(x,a,b,t)\} \le \min_{b \in B} \max_{a \in A} \{\langle \mathbf{f}(x,a,b,t), p\rangle + L(x,a,b,t)\}$$

and in most cases the inequality is strict, and thus the lower and upper value functions are different. But when the above is an equality, then the game is said to satisfy the *minimax conditions*, also called *Isaac's condition*, and we have $\varphi^- = \varphi^+$, and we say the game has *value*.

Our examples will focus on differential games which satisfy the minimax condition, and will thus have value.

In differential games, we usually run into nonconvex Hamiltonians, so it is the Hopf formula (2.3) that is used the most.

3. Splitting algorithms from optimization

Here we review a couple of splitting algorithms from optimization.

3.1. ADMM (Alternating method of multipliers).

ADMM [2], which is also known as Split-Bregman [21], is an optimization method to solve problems of the following form:

$$\begin{aligned} \min_{x,z\in X} \quad & f(x)+g(z) \\ \text{subject to} \quad & Ax+Bz=c \end{aligned}$$

where X is a finite-dimensional real vector space equipped with an inner product $\langle\cdot,\cdot\rangle$, and $f:X\to\mathbb{R}$ and $g:X\to\mathbb{R}$ are proper, convex, lower semicontinuous functions. We also have that A and B are continuous linear operators (e.g. matrices), with c being a fixed element in X. Now we form the *augmented Lagrangian* of the above problem:

$$L_\rho(x,z,y)=f(x)+g(z)+\langle y,Ax+Bz-c\rangle+\frac{\rho}{2}\|Ax+Bz-c\|_2^2.$$

Then we alternately minimize:

$$\begin{cases} x^{k+1} = \arg\min_x L_\rho(x,z^k,y^k) \\ z^{k+1} = \arg\min_z L_\rho(x^{k+1},z,y^k) \\ y^{k+1} = y^k+\rho(Ax^{k+1}+Bz^{k+1}-c) \end{cases}$$

where in the last step we update the dual variable. Note that the $\arg\min$ expressions are frequently precisely the *proximal operator* [36] of a (not necessarily convex) function. The proximal operator is defined as: Given $f:\mathbb{R}^n\to\mathbb{R}$ a proper l.s.c. function, not necessarily convex, then,

$$(I+\lambda\partial f)^{-1}(v):=\arg\min_x\left\{f(x)+\frac{1}{2\lambda}\|x-v\|_2^2\right\}. \tag{3.1}$$

The proximal of f with step-size λ is also denoted $\operatorname{prox}_{\lambda f}(\cdot)$.

3.2. PDHG (Primal-dual hybrid gradient).

The PDHG algorithm [14, 48], which also goes by the name Chambolle-Pock [3], attempts to solve problems of the form

$$\min_{x\in X} \quad f(Ax)+g(x)$$

where we make similar assumptions on X, f, g, and A as we did for ADMM. PDHG takes the Lagrangian dual formulation of the above problem and seeks to find a saddle point of the following problem:

$$\min_{x\in X}\max_{y\in Y}\langle Ax,y\rangle+g(x)-f^*(y)$$

where $f^*(y)=\sup_{x\in X}\{\langle x,y\rangle-f(y)\}$ is the *convex conjugate* of f. PDHG is also an alternating minimization technique that makes use of proximal operators. The updates are:

$$\begin{cases} y^{k+1} = (I+\sigma\partial f^*)^{-1}(y^k+\sigma A\bar{x}^k) \\ x^{k+1} = (I+\tau\partial g)^{-1}(x^k-\sigma A^*y^{k+1}) \\ \bar{x}^{k+1} = x^{k+1}+\theta(x^{k+1}-x^k). \end{cases}$$

where $\sigma,\tau>0$ are such that $\sigma\tau\|A\|^2<1$, and $\theta\in[0,1]$, although $\theta=1$ seems to work best in practice.

4. The generalized Lax and Hopf formulas

A recent result by Y.T. Chow et al. [8] gives a conjectured generalization to the Lax and Hopf formulas. Given a Hamilton-Jacobi equation,

$$\begin{cases} \partial_t\varphi+H(x,\nabla_x\varphi(x,t),t) = 0, \quad \text{in } \mathbb{R}^d\times(0,\infty), \\ \varphi(x,0) = g(x). \end{cases}$$

we have that when $H(x,p,t)$ is smooth, and convex with respect to p, and possibly under some more mild conditions, we have

$$\varphi(x,t)=\min_{v\in\mathbb{R}^d}\left\{g(\mathbf{x}(0))+\int_0^t \mathbf{p}(s)\cdot\nabla_p H(\mathbf{x}(s),\mathbf{p}(s),s)-H(\mathbf{x}(s),\mathbf{p}(s),s)\,ds\right\}$$
$$\text{where}\begin{cases}\dot{\mathbf{x}}(s) &=\nabla_p H(\mathbf{x}(s),\mathbf{p}(s))\\ \dot{\mathbf{p}}(s) &=-\nabla_x H(\mathbf{x}(s),\mathbf{p}(s))\\ \mathbf{x}(t) &=x\\ \mathbf{p}(t) &=v\end{cases}$$

where $\mathbf{x}$ and $\mathbf{p}$ are the characteristics of the PDE. The expression in the bottom braces are ODEs which $\mathbf{x}(\cdot)$ and $\mathbf{p}(\cdot)$ satisfy.

And when we move the convexity onto g, i.e. when $H(x,p,t)$ is smooth and g is convex, then

$$\varphi(x,t)=\sup_{v\in\mathbb{R}^d}\left\{-g^\star(\mathbf{p}(0))+x\cdot v+\int_0^t \mathbf{x}(s)\cdot\nabla_x H(\mathbf{x}(s),\mathbf{p}(s),s)-H(\mathbf{x}(s),\mathbf{p}(s),s)\,ds\right\}$$
$$\text{where}\begin{cases}\dot{\mathbf{x}}(s) &=\nabla_p H(\mathbf{x}(s),\mathbf{p}(s))\\ \dot{\mathbf{p}}(s) &=-\nabla_x H(\mathbf{x}(s),\mathbf{p}(s))\\ \mathbf{x}(t) &=x\\ \mathbf{p}(t) &=v.\end{cases}$$

Chow et al. used coordinate descent with multiple initial guesses to perform the optimization. They do this by first making an initial guess for $v\in\mathbb{R}^d$, then they compute the ODEs, and then compute the value of the objective, i.e. the first lines of the two formulas. Then they re-adjust one coordinate $v\in\mathbb{R}^d$ and repeat. Details can be found in their paper [8].

4.1. Discretizing the generalized Lax and Hopf formulas for optimal control. In order to derive the generalized Lax and Hopf formulas, we can first discretize the value function of the optimal control problem (2.1) and (2.2). Before we begin, we note that we are merely making formal calculations, much in the spirit of E. Hopf in his seminal paper where he derived the classical Hopf formula [24]. This is the procedure followed in [8]: We have that the value function equals

$$\varphi(x,t)=\min_{\mathbf{x}(\cdot),\mathbf{u}(\cdot)}\left\{g(\mathbf{x}(0))+\int_0^t L(\mathbf{x}(s),\mathbf{u}(s),s)\,ds\right\}$$

where $\mathbf{x}(\cdot)$ and $\mathbf{u}(s)$ satisfy the ODE

$$\begin{cases}\dot{\mathbf{x}}(s)=\mathbf{f}(\mathbf{x}(s),\mathbf{u}(s),s), \quad 0<s<t\\ \mathbf{x}(t)=x\end{cases}$$

Two notes: (1) we are formulating our optimal control problem "backwards in time" so that we end up with an *initial-valued* HJ PDE, and (2) here (x,t) are fixed points where we want to compute the HJE.

We discretize the time domain such that

$$0 < s_1 < s_2 < \cdots < s_N = t,$$

and we set $x_j = \mathbf{x}(s_j)$ and $u_j = \mathbf{u}(s_j)$. Note that in our numerical examples, we make a uniform discretization of the time domain.

Now we use the backward Euler discretization of the ODE and set $x_N = x$ to obtain the optimization problem

$$\min_{\{x_j\}_{j=0}^N, \{u_j\}_{j=1}^N} \left\{ g(x_0) + \delta \sum_{j=1}^N L(x_j,u_j,s_j) \,|\, \{x_j - x_{j-1} = \delta f(x_j,u_j,s_j)\}_{j=1}^N, x_N = x \right\}.$$

As is usual in constrained optimization problems, we compute the Lagrangian function (i.e. Lagrange multipliers) to get:

$$g(x_0) + \delta \sum_{j=1}^N L(x_j,u_j,s_j) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} - \delta f(x_j,u_j,s_j)\rangle + \langle p_N, x - x_N\rangle. \tag{4.1}$$

Note that the constraint $x_N = x$ is trivially unneeded in the Lagrangian function. Then we minimize over $\{x_j\}_{j=0}^N$ and $\{u_j\}_{j=1}^N$, while maximizing over $\{p_j\}_{j=1}^N$ to obtain the expression,

$$\max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^{N-1}} \min_{\{u_j\}_{j=1}^N} \left\{ g(x_0) + \delta \sum_{j=1}^N L(x_j,u_j,s_j) \right.$$
$$\left. + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} - \delta f(x_j,u_j,s_j)\rangle + \langle p_N, x - x_N\rangle \right\}.$$

After moving the minimum over $\{u_j\}_{j=1}^N$ inside, we get

$$\max_{\{p_j\}_{j=1}^N} \min_{\{x_k\}_{j=0}^N} \left\{ g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1}\rangle + \langle p_N, x - x_N\rangle \right.$$
$$\left. + \delta \sum_{j=1}^N \min_{u_j} \{L(x_j,u_j,s_j) - \langle p_j, f(x_j,u_j,s_j)\rangle\} \right\}$$
$$= \max_{\{p_j\}_{j=1}^N} \min_{\{x_k\}_{j=0}^N} \left\{ g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1}\rangle + \langle p_N, x - x_N\rangle - \delta \sum_{j=1}^N H(x_j,p_j,s_j) \right\}.$$

After the above step, we have now been able to remove a numerical optimization in u by using the definition of the Hamiltonian. This considerably simplifies the problem, and reduces the dimensionality of the optimization.

We note that we need $p_N = 0$ in order for the maximization/minimization to not be infinite. And thus, we can remove the minimization with respect to x^N and we get

$$\varphi(x,t) \approx \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^N} \left\{ g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1}\rangle - \delta \sum_{j=1}^N H(x_j,p_j,s_j) \right\}. \tag{4.2}$$

We can do a similar analysis using the forward Euler discretization to obtain,

$$\varphi(x,t) \approx \max_{\{p_j\}_{j=0}^{N-1}} \min_{\{x_j\}_{j=0}^{N-1}} \left\{ g(x_0) + \sum_{j=0}^{N-1} \langle p_j, x_{j+1} - x_j \rangle - \delta \sum_{j=0}^{N-1} H(x_j, p_j, s_j) \right\} \tag{4.3}$$

but this latter expression has the disadvantage of coupling the g and H with respect to x_0, as they both depend on x_0. Although this could actually be an advantage as one may have H acting as a regularizer to g.

In order to obtain the discretized version of the generalized Hopf formula, we start with the Lax formula with backward Euler (4.2), and use the linear term $\langle p_1, x_0 \rangle$ and compute the convex conjugate; the calculation goes as follows:

$$\begin{aligned}
&\max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=0}^{N}} \left\{ g(x_0) + \sum_{j=1}^{N-1} \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\} \\
&= \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \Bigg\{ \min_{x_0} \{ g(x_0) - \langle p_1, x_0 \rangle \} + \langle p_1, x_1 \rangle \\
&\qquad\qquad + \sum_{j=2}^{N-1} \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \Bigg\} \\
&= \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \left\{ -g^*(p_1) + \langle p_1, x_1 \rangle + \sum_{j=2}^{N-1} \langle p_j, x_j - x_{j-1} \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\} \\
&= \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \left\{ -g^*(p_1) + \langle p_N, x \rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\}
\end{aligned}$$

where in the last equality, we performed a summation-by-parts and also used $x_N = x$. So we have the discretized version of the Hopf formula:

$$\varphi(x,t) \approx \max_{\{p_j\}_{j=1}^{N}} \min_{\{x_j\}_{j=1}^{N}} \left\{ -g^*(p_1) + \langle p_N, x \rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j \rangle - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\}. \tag{4.4}$$

Note that it is a bit harder to perform the optimization when we approximate the ODE dynamics with forward Euler because we then must compute the convex conjugate of the sum $g(\cdot) + H(\cdot, p_0, s_0)$, which can be more complicated.

4.2. Discretizing the generalized Lax and Hopf formulas for differential games. Again following the procedure of [8], we have a conjectured generalization to the Lax and Hopf formulas for differential games, which we will discretize. Before we give the calculation, we qualify that, in the spirit of E. Hopf when he computed the Hopf formula in his seminal paper [24], these calculations are merely formal:

Given a two-person, zero-sum differential game with value (i.e., it satisfies the Isaacs condition so that the minmax Hamiltonian and maxmin Hamilton are equal, see Section 2.2), with given $x \in \mathbb{R}^{d_1}$ and $y \in \mathbb{R}^{d_2}$, and $t \in (0, \infty)$, and with dynamics

$$\begin{cases} \begin{pmatrix} \dot{\mathbf{x}}(s) \\ \dot{\mathbf{y}}(s) \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \\ \mathbf{f}_2(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \end{pmatrix} & 0 < s < t \\ \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \end{cases}$$

we have that the value function satisfies

$$\varphi(x,y,t) = \inf_{\boldsymbol{\alpha}(\cdot),\mathbf{x}(\cdot)} \sup_{\boldsymbol{\beta}(\cdot),\mathbf{y}(\cdot)} \left\{ g(\mathbf{x}(0),\mathbf{y}(0)) + \int_0^t L(\mathbf{x}(s),\mathbf{y}(s),\boldsymbol{\alpha}(s),\boldsymbol{\beta}(s),s)\,ds \right\}. \tag{4.5}$$

Now, we discretize in time and approximate the ODE with backward Euler, and a formal computation gives us,

$$\varphi(x,y,t) \approx \min_{\{\alpha_k\}_{k=1}^N \{x_k\}_{k=0}^N} \max_{\{\beta_k\}_{k=1}^N, \{y_k\}_{k=0}^N} \left\{ g(x_0,y_0) + \delta \sum_{k=1}^N L(x_k,y_k,\alpha_k,\beta_k,s_k) \right\}$$

$$\text{such that} \begin{cases} \begin{pmatrix} x_k - x_{k-1} \\ y_k - y_{k-1} \end{pmatrix} = \begin{pmatrix} f_1(x_k,y_k,\alpha_k,\beta_k,s_k) \\ f_2(x_k,y_k,\alpha_k,\beta_k,s_k) \end{pmatrix}, \quad k = 1,\dots,N \\ \begin{pmatrix} x_N \\ y_N \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}. \end{cases}$$

It is at this point that we want to form the Lagrangian. The only trouble is that the concept of a "Lagrangian" for minimax problems (a.k.a. saddle-point problems) has not been well-examined. But in a paper by [12] (and also in [37]), the authors have a version of a Lagrangian for minimax problems, which we apply to our problem to get,

$$g(x_0,y_0) + \delta \sum_{j=1}^N L(x_j,y_j,\alpha_j,\beta_j,s_j) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} - \delta f_1(x_j,y_j,\alpha_j,\beta_j,s_j) \rangle$$
$$+ \sum_{j=1}^N \langle -q_j, y_j - y_{j-1} - \delta f_2(x_j,y_j,\alpha_j,\beta_j,s_j) \rangle.$$

Then we take the min max to obtain

$$\begin{aligned}
&\min_{\{\alpha_j\}_{j=1}^N, \{x_j\}_{j=0}^N} \max_{\{\beta_j\}_{j=1}^N, \{y_j\}_{j=0}^N} \Bigg\{ g(x_0,y_0) + \delta \sum_{j=1}^N L(x_j,y_j,\alpha_j,\beta_j,s_j) \\
&\qquad + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} - \delta f_1(x_j,y_j,\alpha_j,\beta_j,s_j) \rangle \\
&\qquad + \sum_{j=1}^N \langle -q_j, y_j - y_{j-1} - \delta f_2(x_j,y_j,\alpha_j,\beta_j,s_j) \rangle \Bigg\} \\
&= \min_{\{x_j\}_{j=0}^N} \max_{\{y_j\}_{j=0}^N} \Bigg\{ g(x_0,y_0) \\
&\qquad + \delta \sum_{j=1}^N \min_{\alpha_j} \max_{\beta_j} \left\{ L(x_j,y_j,\alpha_j,\beta_j,s_j) - \left\langle \begin{pmatrix} p_j \\ -q_j \end{pmatrix}, \begin{pmatrix} f_1(x_j,y_j,\alpha_j,\beta_j,s_j) \\ f_2(x_j,y_j,\alpha_j,\beta_j,s_j) \end{pmatrix} \right\rangle \right\} \\
&\qquad + \sum_{j=1}^N \langle p_j, x_j - x_{j-1} \rangle + \sum_{j=1}^N \langle -q_j, y_j - y_{j-1} \rangle \Bigg\} \\
&= \min_{\{x_j\}_{j=0}^N} \max_{\{y_j\}_{j=0}^N} \Bigg\{ g(x_0,y_0) - \delta \sum_{j=1}^N H(x_j,y_j,p_j,-q_j,s_j)
\end{aligned}$$

$$+\sum_{j=1}^{N}\langle p_j,x_j-x_{j-1}\rangle+\sum_{j=1}^{N}\langle -q_j,y_j-y_{j-1}\rangle\Bigg\}.$$

Now we maximize over $\{p_j\}_{j=1}^N$ and minimize over $\{q_j\}_{j=1}^N$ to obtain,

$$\varphi(x,y,t)\approx \min_{\{q_j\}_{j=1}^N}\max_{\{p_j\}_{j=1}^N}\min_{\{x_j\}_{j=0}^N}\max_{\{y_j\}_{j=0}^N}\Bigg\{g(x_0,y_0)-\delta\sum_{j=1}^{N}H(x_j,y_j,p_j,-q_j,s_j)$$
$$+\sum_{j=1}^{N}\langle p_j,x_j-x_{j-1}\rangle+\sum_{j=1}^{N}\langle -q_j,y_j-y_{j-1}\rangle\Bigg\}$$

and after organizing a bit, we get,

$$\varphi(x,y,t)\approx \min_{\{q_j\}_{j=1}^N}\max_{\{p_j\}_{j=1}^N}\min_{\{x_j\}_{j=0}^N}\max_{\{y_j\}_{j=0}^N}\Bigg\{g(x_0,y_0)+\sum_{j=1}^{N}\left\langle\begin{pmatrix}p_j\\-q_j\end{pmatrix},\begin{pmatrix}x_j-x_{j-1}\\y_j-y_{j-1}\end{pmatrix}\right\rangle$$
$$-\delta\sum_{j=1}^{N}H(x_j,y_j,p_j,-q_j,s_j)\Bigg\}. \quad (4.6)$$

The authors in [12] apply their method to convex-concave differential games, which we also do in Section 6 (see also Section 6.3.3 where we have convex-concave initial conditions).

Note: If we can split $g(x,y)=e(x)+h(y)$, and if e is convex and h is concave, then we may take advantage of e^*, the convex conjugate of e, and h_*, the concave conjugate of h (the formula for the concave conjugate is the same as the convex-conjugate, but you change the sup to an inf) [38], in order to have an analogous Hopf formula:

$$\varphi(x,y,t)\approx \min_{\{q_j\}_{j=1}^N}\max_{\{p_j\}_{j=1}^N}\min_{\{x_j\}_{j=1}^N}\max_{\{y_j\}_{j=1}^N}\Bigg\{-e^*(p_1)-h_*(-q_1)+\left\langle\begin{pmatrix}p_N\\-q_N\end{pmatrix},\begin{pmatrix}x\\y\end{pmatrix}\right\rangle$$
$$+\sum_{j=1}^{N-1}\left\langle\begin{pmatrix}p_j-p_{j+1}\\-(q_j-q_{j+1})\end{pmatrix},\begin{pmatrix}x_j\\y_j\end{pmatrix}\right\rangle-\delta\sum_{j=1}^{N}H(x_j,y_j,p_j,-q_j,s_j)\Bigg\}. \quad (4.7)$$

In some ways the function $e^*(p)+h_*(q)$ may perhaps be called the *convex-concave conjugate* for the convex-concave function $g(x,y)$.

REMARK 4.1. The authors in [12] state that this "minimax Lagrangian", even in the simplest formulation given in their work, is new.

4.3. The advantage of the Hamiltonian for optimization. There is tremendous advantage in having a Hamiltonian. This is because if we want to instead perform optimization of the value function directly, we will be solving for the controls and this requires a constrained optimization technique.

The miraculous advantage of having a Hamiltonian for optimization purposes is it *encodes* information from both the running cost function L, as well as the dynamics $\dot{\mathbf{x}}(s)=\mathbf{f}(\mathbf{x}(s),\mathbf{u}(s),s)$. Thus we are now free to perform *unconstrained optimization.* Instead, if we solve for the value function (2.1) and (2.2) directly, then we need to perform constrained optimization.

Another key additional advantage is that we lower the dimension of the numerical optimization by analytically minimizing over u, and conjuring the Hamiltonian.

4.4. Situations where we don't have a closed-form for the Hamiltonian. Our algorithm currently computes HJ PDEs when we have closed-form expressions of the Hamiltonian H. When this is not the case, then our algorithm will require analytically approximating the Hamiltonian. And computing the proximal of these examples will require some thought. These situations will be examined in future work.

5. The main algorithm: splitting for Hamilton-Jacobi equations

5.1. Splitting for HJE arising from optimal control. Before discussing the algorithms, we note that we do not yet have a proof of convergence nor approximation. This is currently a work-in-progress. But as shown in our numerical results (Section 6), these algorithms seem to agree with classical methods used to solve Hamilton-Jacobi equations.

Taking the Lax formula with backward Euler (4.2) as an expository example, we can organize our problem to look similar to a primal-dual formulation which is attacked by splitting using PDHG. We stack variables and let

- $\tilde{x} = (x_0, x_1, \ldots, x_N)$, and similarly for $\tilde{p}$ and $\tilde{s}$,
- $\tilde{G}(\tilde{x}) = g(x_0)$,
- $\tilde{H}_\delta(\tilde{x}, \tilde{p}, \tilde{s}) = \delta \sum_{k=1}^N H(x_k, u_k, s_k)$,
- D be the difference matrix such that $\langle \tilde{p}, D\tilde{x} \rangle = \sum_{j=1}^{N-1} \langle p_j, x_j - x_{j-1} \rangle$

then our problem looks like:

$$\max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^N} \tilde{G}(\tilde{x}) + \langle \tilde{p}, D\tilde{x} \rangle + \tilde{H}_\delta(\tilde{x}, \tilde{p}, \tilde{s}).$$

This looks similar to the problem that is attacked by PDHG, except for a couple of differences:

- PDHG solves a saddle point problem where the $\tilde{H}_\delta(\tilde{x}, \tilde{p}, \tilde{s})$ term does not depend on $\tilde{x}$ (nor $\tilde{s}$).
- In PDHG, the $\tilde{H}_\delta$ term is the convex conjugate of some function we want to minimize. But in our case, we have

$$H(x, p, s) = \max_u \{\langle f(x, u, s), p \rangle - L(x, u, s)\}$$

  and $f(x, u, s)$ does not even have to be linear. So in some ways, H is a "generalized convex conjugate".

But we perform an alternating minimization technique similar to PDHG and we arrive at our main algorithm for optimal control:

For the Lax formula with backward Euler: Given an $x \in \mathbb{R}^d$ and some time $t \in (0, \infty)$, we set $\delta > 0$ small and let the time-grid size be $N = 1 + t/\delta$ (we set $N = t/\delta$ for Hopf). Then we randomly initialize $\tilde{x} = (x_0, x_1, \ldots, x_N)$ but let $x_N = x$, and we randomly initialize $\tilde{p} = (p_0, p_1, \ldots, p_N)$ but let $p_0 \equiv 0$ as it is not minimized over, but used for computational accounting. And we let $\tilde{z} = \tilde{x}$. Then our algorithm follows the pattern of alternating optimization with quadratic penalty:

$$\begin{cases} \tilde{p}^{k+1} = \arg\max_{\tilde{p}} \left\{ \tilde{G}(\tilde{x}^k) - \tilde{H}_\delta(\tilde{x}^k, \tilde{p}, \tilde{s}) - \frac{1}{2\sigma} \|\tilde{p} - (\tilde{p}^k + D\tilde{z}^k)\|_2^2 \right\} \\ \tilde{x}^{k+1} = \arg\min_{\tilde{x}} \left\{ \tilde{G}(\tilde{x}) - \tilde{H}_\delta(\tilde{x}, \tilde{p}^{k+1}, \tilde{s}) + \frac{1}{2\tau} \|\tilde{x} - (\tilde{x}^k - D^T \tilde{p}^{k+1})\|_2^2 \right\} \\ \tilde{z}^{k+1} = \tilde{x}^{k+1} + \theta(\tilde{x}^{k+1} - \tilde{x}^k). \end{cases}$$

where $\sigma,\tau>0$ are step-sizes with $\sigma\tau\|D\|^2<1$ and $\theta\in[0,1]$ (as suggested in [3]). In our numerical experiments, $\theta=1$ was frequently the best choice and also in practice, we would change the arg max into an arg min. So we have Algorithm 1.

Algorithm 1 Splitting for HJE for optimal control, Lax with backward Euler

```
Given: $x_{\text{target}}\in\mathbb{R}^d$ and time $t\in(0,\infty)$.
Initialize: $\delta>0$ and set $N=1+t/\delta$. And randomly initialize $\tilde{x}^0$ and $\tilde{p}^0$, but with $x_N^0\equiv x_{\text{target}}$. And set $\tilde{z}^0=\tilde{x}^0$. Also choose $\sigma,\tau$ such that $\sigma\tau\|D\|^2<1$ and $\theta\in[0,1]$.
while tolerance criteria large do
    $\tilde{p}^{k+1}=\arg\min_{\tilde{p}}\left\{-\tilde{G}(\tilde{x}^k)+\tilde{H}_\delta(\tilde{x}^k,\tilde{p},\tilde{s})+\frac{1}{2\sigma}\|\tilde{p}-(\tilde{p}^k+\sigma D\tilde{z}^k)\|_2^2\right\}$
    $\tilde{x}^{k+1}=\arg\min_{\tilde{x}}\left\{\tilde{G}(\tilde{x})-\tilde{H}_\delta(\tilde{x},\tilde{p}^{k+1},\tilde{s})+\frac{1}{2\tau}\|\tilde{x}-(\tilde{x}^k-\tau D^T\tilde{p}^{k+1})\|_2^2\right\}$
    $\tilde{z}^{k+1}=\tilde{x}^{k+1}+\theta(\tilde{x}^{k+1}-\tilde{x}^k)$
end while
fval $=g(x_0)+\sum_{j=1}^N\langle p_j,x_j-x_{j-1}\rangle-\delta\sum_{j=1}^N H(x_j,u_j,s_j)$
return fval
```

And a similar algorithm will be obtained when we use a forward Euler discretization Equation (4.3) for the ODE dynamics. We can obtain better accuracy if we average the backward Euler and forward Euler approximations for the ODEs, which is reminiscent of the trapezoidal approximation having better accuracy as it is the average of the forward and backward Euler.

We also have the Hopf formulation: Let,

- $\tilde{G}^*(\tilde{p})=(g^*(p_1),0,\ldots,0)$, and
- let D be the difference matrix such that $\langle D\tilde{p},\tilde{x}\rangle=\langle p_N,x\rangle+\sum_{k=1}^{N-1}\langle p_k-p_{k+1},x_k\rangle$ (so this one differs from Algorithm 1 as it acts on $\tilde{p}$)

then we have Algorithm 2.

Algorithm 2 Splitting for HJE for optimal control, Hopf (with backward Euler)

```
Given: $x_{\text{target}}\in\mathbb{R}^d$ and time $t\in(0,\infty)$.
Initialize: $\delta>0$ and set $N=t/\delta$. And randomly initialize $\tilde{x}^0$ and $\tilde{p}^0$, but with $x_N^0\equiv x_{\text{target}}$. And set $\tilde{z}^0=\tilde{x}^0$. Also choose $\sigma,\tau$ such that $\sigma\tau\|D\|^2<1$ and $\theta\in[0,1]$.
while tolerance criteria large do
    $\tilde{p}^{k+1}=\arg\min_{\tilde{p}}\left\{\tilde{G}^*(\tilde{p}^k)+\tilde{H}_\delta(\tilde{x}^k,\tilde{p},\tilde{s})+\frac{1}{2\sigma}\|\tilde{p}-(\tilde{p}^k+\sigma D^T\tilde{z}^k)\|_2^2\right\}$
    $\tilde{x}^{k+1}=\arg\min_{\tilde{x}}\left\{-\tilde{G}^*(\tilde{p})-\tilde{H}_\delta(\tilde{x},\tilde{p}^{k+1},\tilde{s})+\frac{1}{2\tau}\|\tilde{x}-(\tilde{x}^k-\tau D\tilde{p}^{k+1})\|_2^2\right\}$
    $\tilde{z}^{k+1}=\tilde{x}^{k+1}+\theta(\tilde{x}^{k+1}-\tilde{x}^k)$
end while
fval $=-g^*(p_1)+\langle p_N,x_{\text{target}}\rangle+\sum_{j=1}^{N-1}\langle p_j-p_{j+1},x_j\rangle-\delta\sum_{j=1}^N H(x_j,p_j,s_j)$
return fval
```

5.1.1. When to use the Hopf formula. We make the observation that the Lax formula is suitable (i.e. converges) when we have a convex Hamiltonian in p which is also bounded below in p (or satisfies a coercivity condition, see [16]); if we want a convex Hamiltonian that is not bounded in p, then we must use Hopf in this case.

See Section 5.3 on how to perform the argmin/argmax in each iteration.

5.2. Splitting for HJE arising from differential games. For differential games, we use a similar algorithm to the optimal control case. We take the discretized version of the cost function (4.5) and perform an alternating minimization technique inspired by PDHG, but applied to minimax problems. Using the same notation as in Algorithm 1 and Algorithm 2, and the same D matrix as in Algorithm 1, we have the algorithm for differential games in Algorithm 3.

Algorithm 3 Splitting for HJE for differential games, Lax

```
Given: $(x_{\text{target}}, y_{\text{target}}) \in \mathbb{R}^d$ and time $t \in (0,\infty)$.
Initialize: $\delta > 0$ and set $N = 1 + t/\delta$. And randomly set $\tilde{x}^0$, $\tilde{y}^0$, $\tilde{p}^0$, and $\tilde{q}^0$, but with $x_N^0 \equiv x_{\text{target}}$ and $y_N^0 \equiv y_{\text{target}}$. And set $(\bar{\tilde{x}}^0, \bar{\tilde{y}}^0) = (\tilde{x}^0, \tilde{y}^0)$. Also choose $\sigma, \tau$ such that $\sigma\tau\|D\|^2 < 1$ and $\theta \in [0,1]$.
while tolerance criteria large do
    $\tilde{p}^{k+1} = \arg\max_{\tilde{p}} \left\{ \tilde{G}(\tilde{x}^k, \tilde{y}^k) - \tilde{H}_\delta(\tilde{x}^k, \tilde{y}^k, \tilde{p}, -\tilde{q}^k, \tilde{s}^k) - \frac{1}{2\sigma}\|\tilde{p} - (\tilde{p}^k + \sigma D \bar{\tilde{x}}^k)\|_2^2 \right\}$
    $\tilde{q}^{k+1} = \arg\min_{\tilde{q}} \left\{ \tilde{G}(\tilde{x}^k, \tilde{y}^k) - \tilde{H}_\delta(\tilde{x}^k, \tilde{y}^k, \tilde{p}^{k+1}, -\tilde{q}, \tilde{s}^k) + \frac{1}{2\sigma}\|\tilde{q} - (\tilde{q}^k + \sigma D \bar{\tilde{y}}^k)\|_2^2 \right\}$
    $\tilde{x}^{k+1} = \arg\min_{\tilde{x}} \left\{ \tilde{G}(\tilde{x}, \tilde{y}^k) - \tilde{H}_\delta(\tilde{x}, \tilde{y}^k, \tilde{p}^{k+1}, -\tilde{q}^{k+1}, \tilde{s}^k) + \frac{1}{2\tau}\|\tilde{x} - (\tilde{x}^k - \tau D^T \tilde{p}^{k+1})\|_2^2 \right\}$
    $\tilde{y}^{k+1} = \arg\max_{\tilde{y}} \left\{ \tilde{G}(\tilde{x}^{k+1}, \tilde{y}) - \tilde{H}_\delta(\tilde{x}^{k+1}, \tilde{y}, \tilde{p}^{k+1}, -\tilde{q}^{k+1}, \tilde{s}^k) \right.$
                  $\left. - \frac{1}{2\tau}\|\tilde{y} - (\tilde{y}^k - \tau D^T \tilde{q}^{k+1})\|_2^2 \right\}$
    $\begin{pmatrix} \bar{\tilde{x}}^{k+1} \\ \bar{\tilde{y}}^{k+1} \end{pmatrix} = \begin{pmatrix} \tilde{x}^{k+1} \\ \tilde{y}^{k+1} \end{pmatrix} + \theta \left( \begin{pmatrix} \tilde{x}^{k+1} \\ \tilde{y}^{k+1} \end{pmatrix} - \begin{pmatrix} \tilde{x}^{k} \\ \tilde{y}^{k} \end{pmatrix} \right).$
end while
fval $= g(x_0, y_0) + \sum_{j=1}^N \left\langle \begin{pmatrix} p_j \\ -q_j \end{pmatrix}, \begin{pmatrix} x_j - x_{j-1} \\ y_j - y_{j-1} \end{pmatrix} \right\rangle - \delta \sum_{j=1}^N H(x_j, y_j, p_j, -q_j, s_j)$
return fval
```

If we have $\tilde{G}(x,y) = \tilde{E}(x) + \tilde{H}(y)$ where E is convex and H is concave, then we may make use of convex-conjugates and concave-conjugates (see (4.7)) to obtain an analogous Hopf formula as in Algorithm 2, but for differential games. Here D is the same difference matrix as in the Hopf case, Algorithm 2. This is Algorithm 4.

5.3. Remarks on how to perform the argmin/argmax in each iteration. In each iteration for the above algorithms, we have an optimization problem when updating $\tilde{x}^{k+1}$ ($\tilde{y}^{k+1}$) or $\tilde{p}^{k+1}$ ($\tilde{q}^{k+1}$). In some of our experiments, the optimization turned into a closed-form proximal expression (mainly when updating $\tilde{p}^{k+1}/\tilde{q}^{k+1}$, or we were able to make use of one step of gradient descent of the objective (mainly when updating $\tilde{x}^{k+1}$ or $\tilde{y}^{k+1}$ or when $\tilde{G} = g$ is involved). As an example, one way to update the Lax formula for optimal control (Algorithm 1) using a backward Euler discretization is

$$\begin{cases} \tilde{p}^{k+1} &= \operatorname{prox}_{\sigma \tilde{H}_\delta(\bar{\tilde{x}}^k, \cdot)}(\tilde{p}^k + \sigma D \bar{\tilde{x}}^k) \\ \\ \tilde{x}^{k+1} &= \tilde{x}^k - \tau D^T \tilde{p}^{k+1} - \tau \nabla_{\tilde{x}} \tilde{G}(\tilde{x}^k) + \tau \nabla_{\tilde{x}} \tilde{H}_\delta(\tilde{x}^k, \tilde{p}^{k+1}, \tilde{s}) \\ \\ \bar{\tilde{x}}^{k+1} &= \tilde{x}^{k+1} + \theta(\tilde{x}^{k+1} - \tilde{x}^k). \end{cases}$$

Algorithm 4 Splitting for HJE for differential games, Hopf (for separable convex-concave initial conditions)

```
Given: $(x_{\text{target}},y_{\text{target}})\in\mathbb{R}^d$ and time $t\in(0,\infty)$.
Initialize: $\delta>0$ and set $N=t/\delta$. And randomly set $\tilde{x}^0$, $\tilde{y}^0$, $\tilde{p}^0$, and $\tilde{q}^0$, but with $x_N^0\equiv x_{\text{target}}$ and $y_N^0\equiv y_{\text{target}}$. And set $(\bar{\tilde{x}}^0,\bar{\tilde{y}}^0)=(\tilde{x}^0,\tilde{y}^0)$. Also choose $\sigma,\tau$ such that $\sigma\tau\|D\|^2<1$ and $\theta\in[0,1]$.
while tolerance criteria large do
    $\tilde{p}^{k+1}=\arg\max_{\tilde{p}}\Big\{-\tilde{E}^*(p_1)-\tilde{H}_*(-q_1)-\tilde{H}_\delta(\tilde{x}^k,\tilde{y}^k,\tilde{p},-\tilde{q}^k,\tilde{s}^k) - \frac{1}{2\sigma}\|\tilde{p}-(\tilde{p}^k+\sigma D^T\bar{\tilde{x}}^k)\|_2^2\}$
    $\tilde{q}^{k+1}=\arg\min_{\tilde{q}}\Big\{-\tilde{E}^*(p_1)-\tilde{H}_*(-q_1)-\tilde{H}_\delta(\tilde{x}^k,\tilde{y}^k,\tilde{p}^{k+1},-\tilde{q},\tilde{s}^k) + \frac{1}{2\sigma}\|\tilde{q}-(\tilde{q}^k+\sigma D^T\bar{\tilde{y}}^k)\|_2^2\}$
    $\tilde{x}^{k+1}=\arg\min_{\tilde{x}}\Big\{-\tilde{E}^*(p_1)-\tilde{H}_*(-q_1)-\tilde{H}_\delta(\tilde{x},\tilde{y}^k,\tilde{p}^{k+1},-\tilde{q}^{k+1},\tilde{s}^k) + \frac{1}{2\tau}\|\tilde{x}-(\tilde{x}^k-\tau D\tilde{p}^{k+1})\|_2^2\}$
    $\tilde{y}^{k+1}=\arg\max_{\tilde{y}}\Big\{-\tilde{E}^*(p_1)-\tilde{H}_*(-q_1)-\tilde{H}_\delta(\tilde{x}^{k+1},\tilde{y},\tilde{p}^{k+1},-\tilde{q}^{k+1},\tilde{s}^k) - \frac{1}{2\tau}\|\tilde{y}-(\tilde{y}^k-\tau D\tilde{q}^{k+1})\|_2^2\}$
    $\begin{pmatrix}\bar{\tilde{x}}^{k+1}\\ \bar{\tilde{y}}^{k+1}\end{pmatrix}=\begin{pmatrix}\tilde{x}^{k+1}\\ \tilde{y}^{k+1}\end{pmatrix}+\theta\left(\begin{pmatrix}\tilde{x}^{k+1}\\ \tilde{y}^{k+1}\end{pmatrix}-\begin{pmatrix}\tilde{x}^{k}\\ \tilde{y}^{k}\end{pmatrix}\right)$.
end while
fval $=-e^*(p_1)-h_*(-q_1)+\left\langle\begin{pmatrix}p_N\\ -q_N\end{pmatrix},\begin{pmatrix}x\\ y\end{pmatrix}\right\rangle+\sum_{j=1}^{N-1}\left\langle\begin{pmatrix}p_j-p_{j+1}\\ -(q_j-q_{j+1})\end{pmatrix},\begin{pmatrix}x_j\\ y_j\end{pmatrix}\right\rangle-\delta\sum_{j=1}^{N}H(x_j,y_j,p_j,-q_j,s_j)$
return fval
```

and one way to update the Hopf formula for optimal control (Algorithm 2) is,

$$\begin{cases}\tilde{p}^{k+1} &= \operatorname{prox}_{\sigma\tilde{H}_\delta(\bar{\tilde{x}}^k,\cdot)}(\tilde{p}^k+\sigma D\bar{\tilde{x}}^k-\sigma\nabla g^*(p_1))\\ \tilde{x}^{k+1} &= \tilde{x}^k-\tau D^T\tilde{p}^{k+1}+\tau\nabla_{\tilde{x}}\tilde{H}_\delta(\tilde{x}^k,\tilde{p}^{k+1})\\ \bar{\tilde{x}}^{k+1} &= \tilde{x}^{k+1}+\theta(\tilde{x}^{k+1}-\tilde{x}^k).\end{cases}$$

where we see the update for $\tilde{p}^{k+1}$ is a *proximal gradient* update [36] (Section 4.2).

There are many combinations we can use, but the intuition is to take a gradient step on the smooth part (usually the x part), and a proximal step on the non-smooth part (usually the p part). And if we have a sum of a smooth part and a non-smooth part, we can mix a gradient step with a proximal step (i.e. a proximal gradient step) as we have done above.

Currently, we do not have a theory of convergence, but these algorithms work experimentally. This will be left for future work.

5.4. Computation of characteristic curves/optimal trajectories. A benefit from the new algorithm is that alongside computing solutions at each point, it also allows us to directly compute trajectories/characteristic curves of the Hamilton-Jacobi equation at each point. In fact, our algorithm is a hybrid of the direct collocation method (a direct method) [46] and Pontryagin's maximum principle (an indirect method) [22].

We give some examples of characteristic curves/optimal trajectories in our numerical results (Section 6).

5.5. The advantage of splitting over coordinate descent. The advantage of these methods over coordinate descent used previously in Chow et. al. [8] is not only its speed, but it also does not seem to require as many multiple initial guesses for nonconvex optimization. In our numerical experiments in Section 6, we only used a single initial guess in all our examples. And for most examples in our experiments in Section 6, it only requires one guess.

It also has the advantage that one can apply the method to *non-smooth* problems, as opposed to coordinate-descent, where one takes numerical gradients.

And practically, splitting is more straightforward to implement than coordinate descent, where we would require divided differences to numerically compute the gradients, and we also have available to us the multitude of splitting techniques from the optimization literature, such as ADMM and Douglas-Rachford splitting to name a few.

5.6. A remark on the connection between Hamilton-Jacobi equations and optimization, and the implications on future optimization methods. The relationships between Hamilton-Jacobi equations and optimization have been noted in the literature [39,40]. More recently, there has been a connection between deep learning optimization and HJE [5]. More concretely, there is a straight-forward connection between Hamilton-Jacobi equations and the proximal operator (which can be interpreted as implicit gradient descent):

Given a function $f(\cdot)$, the proximal operator of f is

$$(I+\lambda\partial f)^{-1}(v) := \arg\min_x \left\{ f(x) + \frac{1}{2\lambda}\|x-v\|_2^2 \right\}.$$

If we change the $\arg\min_x$ into a $\min_x$, then we get the familiar Lax (a.k.a. Hopf-Lax) formula for the Hamilton-Jacobi equation with $H(x) = \frac{1}{2}\|x\|_2^2$, and for $t=\lambda$. So in this way, the generalized Lax and Hopf formulas can be viewed as a generalization of the proximal operator.

Also, the proximal operator, i.e. the $\arg\min_x$ operator, is featured heavily in our algorithms. In classical PDHG, the primal variable x and the dual variable p are decoupled. But for our algorithms, the coupling is in the form of a state-dependent Hamiltonian. This coupling of the primal and dual variables can have implications on future optimization methods, as we can then attempt various coupling functions, i.e. Hamiltonians, and examine their effectiveness in general optimization techniques.

We also reiterate that our algorithms have been able to perform nonconvex optimization without as many multiple initial guesses as in other algorithms such as coordinate descent. In fact, in all our examples in Section 6, we only used a single initial guess. So we feel a deeper theoretical examination of these algorithms will likely be beneficial to the theory and literature of nonconvex optimization methods.

6. Numerical results

Here we present numerical examples using our algorithm. The computations were run on a laptop with an Intel i7-4900MQ 2.90GHz×4 Haswell Core processor, of which only one core processor was utilized for computations. The computations for the Eikonal equation and the difference of norms were computed in C++, while the (unnamed) Isaacs example and the quadcopter were computed in MATLAB, version R2017b.

For initializations, we initialized $\tilde{x}^0 = (x_0^0, x_1^0, \ldots, x_N^0)$ to be such that each x_i^0 (for $i=0,\ldots,N-1$) is a random point close to x_{target}, and we let $x_N^0 \equiv x_{\text{target}}$. In particular, for $\tilde{x}^0$ each coordinate, except for x_N^0, was randomly initialized so that $\|\tilde{x}^0 - (x_{\text{target}}, x_{\text{target}}, \ldots, x_{\text{target}})\|_\infty \leq 0.1$. We chose $\tilde{p}^0$ to be a random vector close to the origin so that $\|\tilde{p}^0 - 0\|_\infty \leq 0.1$.

We chose $\theta=1$ in all cases.

The PDHG step-sizes σ and τ, and the time-step size δ all varied for each example.

The error tolerance for all optimal control examples were chosen such that primal and dual variables satisfy $\|x^{k+1}-x^k\|_2^2<10^{-8}$ and $\|p^{k+1}-p^k\|_2^2<10^{-8}$. The error tolerance for all the differential games examples were also similarly $\|(x^{k+1},y^{k+1})-(x^k,y^k)\|_2^2<10^{-8}$ and $\|(p^{k+1},q^{k+1})-(p^k,q^k)\|_2^2<10^{-8}$, although we had to slightly modify our stopping criteria here.

For the difference of norms example, if the algorithm reached some max count, then we would examine the value function and stop the algorithm when the value of the value function for consecutive iterations reached a difference below some tolerance.

For the (unnamed) Isaacs example with fully convex initial conditions, we chose the error to be such that the relative error of the value function of consecutive iterations was less than 10^{-6}, i.e. $\left\|\frac{\mathrm{fval}^{k+1}-\mathrm{fval}^k}{\min(\|\mathrm{fval}^{k+1}\|,1)}\right\|<10^{-6}$. This example turned out to be the harshest on our algorithm.

In all cases, when we derive the Hamiltonian, we are starting from an optimal control with a terminal condition and solving "backwards in time" (see Section 2.1) as this naturally gives us an initial-valued Hamilton-Jacobi PDE.

6.1. State-and-time-dependent Eikonal equation (Optimal control).

6.1.1. Brief background on Eikonal equations. The state-dependent Eikonal equations are HJE with Hamiltonians,

$$H^+(x,p)=c(x)\|p\|, \quad H^-=-c(x)\|p\|$$

where $c(x)>0$ and $\|\cdot\|$ is any norm; in our examples we take the Euclidean norm. We also test a state-and-time-dependent equation of Eikonal type, where $H(x,p,t)=c(x-t)\|p\|$.

They arise from optimal control problems that have dynamics

$$f(x,u)=c(x)u, \quad \text{with } \|u\|\leq 1 \text{ and } c(x)\neq 0 \text{ for all } x$$

and with cost-functional

$$J[u]=g(x(0))+\int_0^t \mathcal{I}_{\leq 1}(u(s))\,ds$$

where $\mathcal{I}_{\leq 1}(\cdot)$ is the indicator function of the unit ball in $\mathbb{R}^n$, i.e. 0 for all points within the unit ball including the boundary, and $+\infty$ for all points outside.

This is a *nonlinear* optimal control example due to the presence of $c(x)$. Also, our algorithm is performing nonconvex optimization (due to the presence of $c(x)$, but also in our negative Eikonal equation example where we are performing minimization with a $-g(x)$ term where g is quadratic).

The Eikonal equation features heavily in the level set method [33, 34], which has made wide-ranging contributions in many fields.

Note the optimization in solving negative Eikonal equation can be obtained by examining the positive Eikonal equation. This is actually a general phenomenon of Hamiltonians that obey $H(x,-p,t)=H(x,p,t)$. This is because if φ solves a HJE with initial data g and Hamiltonian such that $H(x,-p,t)=H(x,p,t)$, then examining $-\varphi$,

$$\left\{\begin{array}{r}(-\varphi)_t+H(x,\nabla(-\varphi),t)=0\\ (-\varphi)(x,0)=-g(x)\end{array}\right\} \quad\Leftrightarrow\quad \left\{\begin{array}{r}\varphi_t-H(x,\nabla\varphi,t)=0\\ \varphi(x,0)=g(x)\end{array}\right\}$$

so we see $-\varphi$ solves the positive Eikonal equation with initial data $-g$ if and only if φ solve the negative Eikonal equation with initial data g. And note that both are viscosity solutions as this computation still holds when we compute the viscous version of the HJE [16] (Section 10.1).

6.1.2. Implementation details for the Eikonal equation.

Here we take

$$c(x) = 1 + 3\exp(-4\|x-(1,1,0,\dots,0)\|_2^2),$$

which is a positive bump function. The initial condition of our HJE PDE is

$$g(x) = -1/2 + (1/2)\langle A^{-1}x, x\rangle, \quad A = \operatorname{diag}(2.5^2, 1, 0.5^2, \dots, 0.5^2).$$

We used the Lax version of our algorithm, Algorithm 1, for all cases in this section. For the $\tilde{x}^{k+1}$-update, we took one step of gradient descent. And for the $\tilde{p}^{k+1}$ update, we took the proximal of the ℓ_2-norm (a.k.a. the *shrink*$_2$ operator); in general the *shrink* operator can be defined for any positively homogenous of degree 1 convex function φ. For the ℓ-1 norm, we have that the *shrink*$_1$ operator (in $\mathbb{R}^n$) can be computed coordinate-wise and the i-th coordinate satisfies,

$$(\text{shrink}_1(x,\lambda))_i = \begin{cases} x_i - \lambda & \text{if } x_i > \lambda \\ 0 & \text{if } |x_i| \le \lambda \\ x_i + \lambda & \text{if } x_i < -\lambda \end{cases}$$

and the *shrink*$_2$ operator also can be computed coordinate-wise and the i-th coordinate satisfies

$$(\text{shrink}_2(x,\lambda))_i = \begin{cases} \frac{x}{\|x\|_2}\max(\|x\|_2 - \lambda, 0) & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases}$$

For the negative Eikonal equation, since in our implementation we computed a positive Eikonal equation with initial data $-g$ and then took the negative, we were able to compute the proximal of the concave quadratic $-g$. We call this taking the *stretch* operator of g (see [6], Section 4.2.2).

For these Eikonal equations, we chose a time step-size of $\delta = 0.02$, and we computed in a $[-3,3]^2$ grid with a mesh size of 0.1 on each axis.

For the positive Eikonal Equation (fig. 6.1), we chose a PDHG step-size that depended on the norm of $\nabla c(x)$. If $\|\nabla c(x)\|_2 > 0.001$, then we took $\sigma = 50$ or else we took $\sigma = 0.5$. And we always took $\tau = 0.25/\sigma$ (the 0.25 comes from $\|D\|_2 = 2 - \varepsilon$ for some small $\varepsilon > 0$, and requiring $\sigma\tau < 1/\|D\|^2$). To compute the times $t = 0.1$, 0.2, 0.3, and 0.4, the computation time averaged to 4.39×10^{-4} seconds per point in C++.

For the negative Eikonal Equation (fig. 6.2), we chose a PDHG step-size of $\sigma = 100$ and $\tau = 0.25/\sigma$. We picked $\theta = 1$ for all cases. For $t = 0.1$, 0.2, 0.3, 0.4, and 0.5, the computation time averaged to 0.0024 seconds per point in C++.

For the negative Eikonal equation in 10 dimensions (fig. 6.3), we computed a 2-dimensional slice in $[-3,3] \times [-3,3] \times \{0\} \times \cdots \times \{0\}$. The time step-size was again $\delta = 0.02$ and we chose $\sigma = 100$, and $\tau = 0.25/\sigma$. For $t = 0.1$, 0.2, 0.3, and 0.4 (0.5 had no level sets) the computation time averaged to 0.004 seconds per point in C++.

We also computed a state-and-time-dependent Eikonal Equation (fig. 6.4) where $H(x,p,t) = c(x - t(-1,1))\|p\|_2$. Here in our specific example, $c(x - t(-1,1))$ represents a bump function moving diagonally in the $(-1,1)$ direction as t increases. The time step-size were the same as in the positive Eikonal case which is reasonably expected because we took the positive Eikonal case and modified it. The computational time for $t=0.1$, 0.2, 0.3, and 0.4 averaged to 5.012×10^{-4} seconds per point in C++.

In these examples, we achieved a speedup of about ten times over coordinate descent used by Chow et. al. [8], and only one initial guess was used. In low dimensions, this problem can be solved with SQP (Sequential Quadratic Programming) on the value function, or using Lax-Friedrichs. We use these methods to check our accuracy and they agree to within 10^{-4} for each point (x,t) when using SQP.

We observe that the two different Eikonal equations required vastly different step-sizes and it is worth examining how to choose step-sizes in a future work. We speculate that the step-sizes may act as CFL conditions. This is a further point of study.

Comparing coordinate descent to our algorithm in MATLAB, we achieve about an 8-10 times speedup used in [8].

The figures 6.1, 6.2, and 6.3, and 6.4 show the zero level sets of the HJE solution.

In Algorithm 5, we give an explicit example of how we performed our algorithm on the negative Eikonal equation.

Algorithm 5 Splitting for the negative Eikonal equation with convex initial data g (note: below we are actually solving the equivalent problem of the positive Eikonal equation with concave initial data $-g$)

```
Given: $x_{\text{target}}\in\mathbb{R}^d$ and time $t\in(0,\infty)$.

while ($\|\tilde{x}^{k+1}-\tilde{x}^k\|_2^2>$ tol or $\|\tilde{p}^{k+1}-\tilde{p}^k\|_2^2>$ tol) and (count $<$ max_count) do
  for $j=1$ to $N$ do
    $p_j^{k+1}=\text{shrink}_2(p_j^k+\sigma(z_j^k-z_{j-1}^k),\sigma\delta c(x^k))$ (proximal)
  end for

  for $j=0$ do
    $x_0^{k+1}=x_0^k-\tau(\underbrace{p_0^{k+1}}_{=0}-p_1^{k+1})-\tau(\nabla(-g))(x_0^k)$ (gradient descent)
  end for
  for $j=1$ to $N-1$ do
    $x_j^{k+1}=x_j^k-\tau(p_j^{k+1}-p_{j-1}^{k+1})-\tau(-\delta(\nabla c)(x_j^k)\|p_j^{k+1}\|_2)$ (gradient descent)
  end for

  for $j=0$ to $N$ do
    $z_j^{k+1}=x_j^{k+1}+\theta(x_j^{k+1}-x_j^k)$ (extrapolation step)
  end for
end while

fval $=-g(x_0)+\sum_{j=1}^N\langle p_j,x_j-x_{j-1}\rangle-\delta\sum_{j=1}^N c(x_j)\|p_j\|_2$
return fval
```

6.1.3. Dimensional scaling of the negative Eikonal equation. Here we examine how Algorithm 1 scales with dimension. We compute the negative Eikonal equation with the same speed c and initial data g as above, and with $\delta=0.02$ and $\sigma=100$ and $\tau=0.25/\sigma$. We computed in a 2-dimensional slice $[-3,3]^2\times\{0\}^{d-2}$, and we computed from $d=10$ to $d=2000$ dimensions. We performed our analysis at time $t=0.2$.

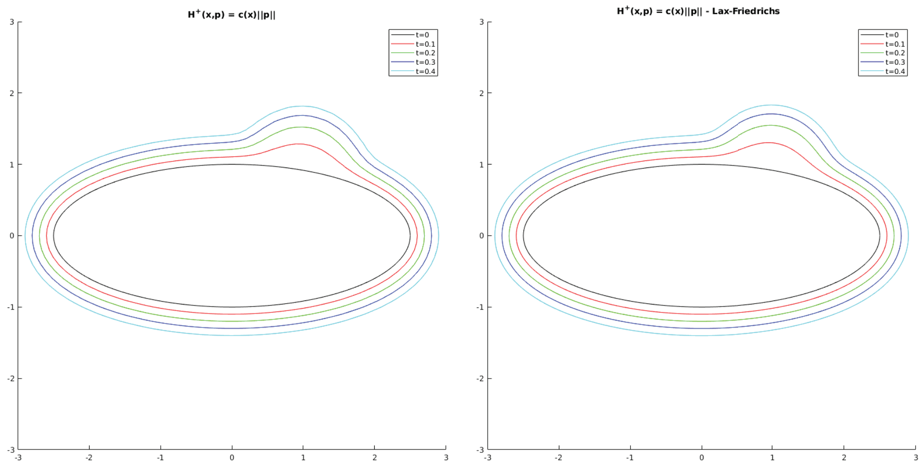

FIG. 6.1. *Eikonal equation with $H^+(x,p)=c(x)\|p\|_2$, in two spatial dimensions. This plot shows the zero level sets of the HJE solution for $t=0.1,0.2,0.3,0.4$. We observe that the zero level sets move outward as time increases. The left figure is computed using our new algorithm, while the right figure is computed using the conventional Lax-Friedrichs method.*

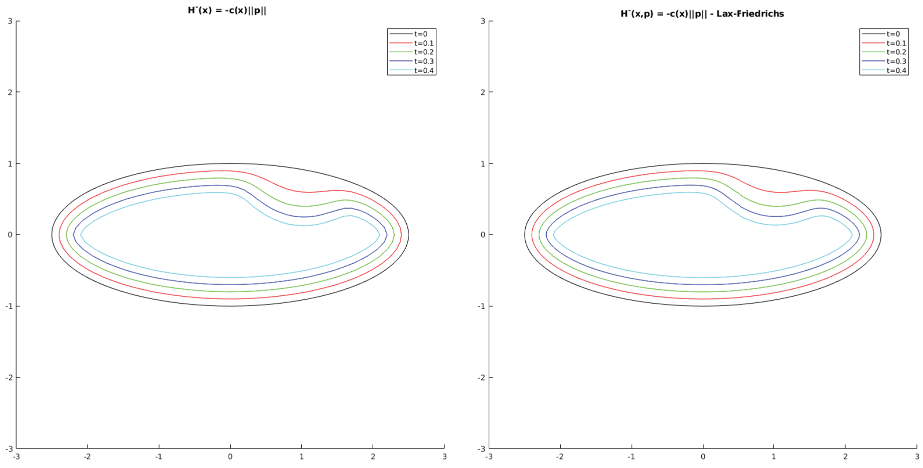

FIG. 6.2. *Eikonal equation with $H^-(x,p)=-c(x)\|p\|_2$, in two spatial dimensions. This plot shows the zero level sets of the HJE solution for $t=0.1,0.2,0.3,0.4$. We observe that the zero level sets move inward as time increases. Left is computed with our new algorithm, while the right is computed using the conventional Lax-Friedrichs method.*

We chose this particular example as this is a nonlinear optimal control problem that requires us to optimize a nonconvex problem.

We used least-squares to fit both a linear function as well as a quadratic function. The coefficients were

$$\begin{aligned}\mathrm{lin}(d) &= (1.14\times 10^{-4})d + 0.0021,\\ \mathrm{quad}(d) &= (-5.99\times 10^{-9})d^2 + (1.27\times 10^{-4})d - 0.00195\end{aligned}$$

As we can see from the equations of the fit, and from fig. 6.5, the quadratic fit has an

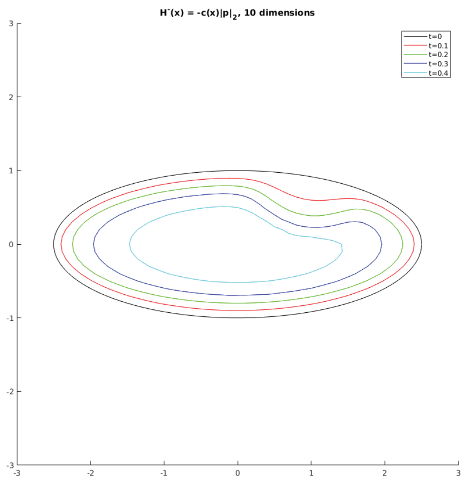

FIG. 6.3. *Eikonal equation with $H^{-}(x,p) = -c(x)\|p\|_2$, in ten spatial dimensions. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3, 0.4$. We observe that the zero level sets move inward as time increases.*

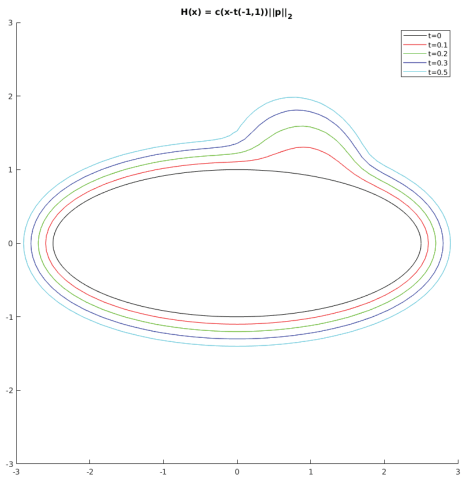

FIG. 6.4. *Eikonal equation with $H(x,p) = c(x - t(-1,1))\|p\|_2$. This plot shows the zero level sets of the HJE solution for $t = 0.1, 0.2, 0.3, 0.4$. We observe that there is a similarity to the positive Eikonal case, but the "bump" is more sheared to the left.*

extremely small quadratic coefficient. Figure 6.5 shows the plot with the linear fit. This computation was done in C++.

We predict that for general problems, the scaling will be polynomial in time.

6.2. Difference of norms (differential games).

6.2.1. From differential games to HJE for the difference of norms. The state-dependent HJE for the difference of norms case arises from the following differential games problem: Given $x \in \mathbb{R}^{d_1}$ and $y \in \mathbb{R}^{d_2}$, and some $t > 0$, we have the following

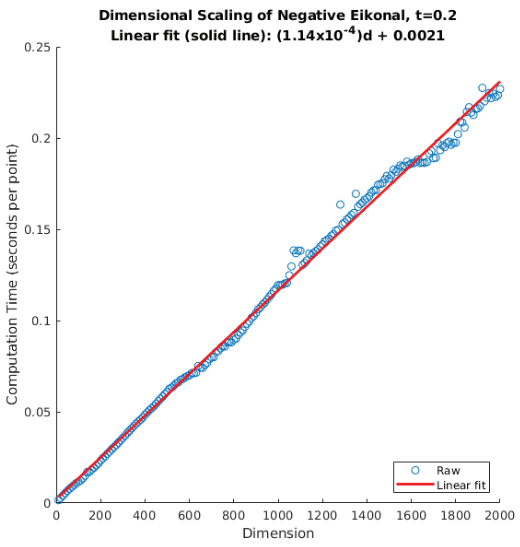

FIG. 6.5. *How our algorithm scales with dimension for the negative Eikonal equation at time $t=0.2$. This is a nonlinear optimal control problem, and the optimization requires us to perform nonconvex optimization. The plot shows a linear fit.*

dynamics

$$\begin{cases} \begin{pmatrix} \dot{\mathbf{x}}(s) \\ \dot{\mathbf{y}}(s) \end{pmatrix} = \begin{pmatrix} c_1(\mathbf{x}(s),\mathbf{y}(s))\boldsymbol{\alpha}(s) \\ c_2(\mathbf{x}(s),\mathbf{y}(s))\boldsymbol{\beta}(s) \end{pmatrix}, \quad 0 \le s \le t \\ \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \\ \boldsymbol{\alpha}(s)\| \le 1, \|\boldsymbol{\beta}(s)\| \le 1, \text{ for all } s, \\ \text{and } c_1, c_2 \text{ are positive functions.} \end{cases}$$

And the cost function is

$$J_{x,t}[\boldsymbol{\alpha},\boldsymbol{\beta}] = g(\mathbf{x}(0),\mathbf{y}(0)) + \int_0^t \mathcal{I}_{\le 1}(\boldsymbol{\alpha}(s)) - \mathcal{I}_{\le 1}(\boldsymbol{\beta}(s))\,ds$$

where $\mathcal{I}_{\le 1}(\cdot)$ is the indicator function of the unit-ball, i.e. it equals 0 for points inside and on the unit-ball, and $+\infty$ for points outside. Our value function is then

$$\varphi(x,y,t) = \inf_{\alpha,\|\alpha\|\le 1} \sup_{\beta,\|\beta\|\le 1} J_{x,t}[\boldsymbol{\alpha},\boldsymbol{\beta}].$$

Then our Hamiltonian becomes,

$$\begin{aligned} H(x,y,p,q) &= \max_{\alpha} \min_{\beta} \left\{ \left\langle \begin{pmatrix} c_1(x,y)\alpha \\ c_2(x,y)\beta \end{pmatrix}, \begin{pmatrix} p \\ q \end{pmatrix} \right\rangle - (I_{\le 1}(\alpha) - I_{\le 1}(\beta)) \right\} \\ &= \max_{\alpha} \{\langle c_1(x,y)\alpha, p_1\rangle - I_{\le 1}(\alpha)\} + \min_{\beta} \{\langle c_2(x,y)\beta, q\rangle + I_{\le 1}(\beta)\} \\ &= c_1(x,y)\|p\|_2 - c_2(x,y)\|q\|_2. \end{aligned}$$

In this case, we have nonlinear dynamics, as well as nonconvex Hamiltonian.

6.2.2. Implementation details for the difference of norms. Here we take,

$$c_1(x) = 1 + 3\exp(-4\|x-(1,1,0,\ldots,0)\|_2^2), \qquad c_2(x) = c_1(-x),$$

and the initial condition is

$$g(x) = -1/2 + (1/2)\langle A^{-1}x, x\rangle, \quad A = \mathrm{diag}(2.5^2, 1, 0.5^2, \ldots, 0.5^2).$$

which is the same initial condition as in our Eikonal equation example above (Section 6.1.2).

For the 2-dimensional case, we used the Hamiltonian,

$$H(x_1,x_2,p_1,p_2) = c(x_1,x_2)\|p_1\|_2 - c(-x_1,-x_2)\|p_2\|_2$$

and for the 7-dimensional case, we used,

$$H(x_1,x_2,\ldots,x_7,p_1,p_{(2,\ldots,7)}) = c(x_1,\ldots,x_7)\|p_1\|_2 - c(-x_1,\ldots,-x_7)\|p_{(2,\ldots,7)}\|_2.$$

where $p_{(2,\ldots,7)} = (p_2, p_3, \ldots, p_7)$.

We compute our solutions in 2 and 7 dimensions. We compute the 2-dimensional case in a $[-3,3]^2$ grid, and we compute the 7-dimensional case in the two dimensional slice $[-3,3]^2 \times \{0\} \times \cdots \times \{0\}$. And we used Algorithm 3.

For the $\tilde{p}^{k+1}$-update, we used the proximal of the ℓ_2-norm (a.k.a. the *shrink*$_2$ operator), and for the $\tilde{x}^{k+1}$-update, we used one step of gradient descent.

We took the time-step as $\delta = 0.02$, and we took the PDHG steps $\sigma = 50$, and $\tau = 0.25/\sigma$. The 0.25 comes from the fact that the PDHG algorithm requires $\sigma\tau\|D\|_2^2 < 1$, and $\|D\|_2 = 2 - \varepsilon$, for some small $\varepsilon > 0$.

The computation was done with a mesh size of $1/12 \approx 0.08333$ in each axis. For the 2-dimensional case, the computation averaged out to 0.0135 seconds per point in C++, and for the 7-dimensional case the computation averaged out to 0.01587 seconds per point in C++. If we compared the algorithms in MATLAB on the same computer, we achieved a 10-20 times speedup compared to coordinate descent used previously by Chow et al. [8]. In order to compare the accuracy of the solution, we present our method compared with the Lax-Friedrichs method for solving the Hamilton-Jacobi equation in fig. 6.6. Our method for the 7-dimensional case is shown in fig. 6.7.

We note that at certain points, the trajectories would oscillate a little for larger times which may be due to the non-convexity and the non-unique optimal trajectories. So when a maximum count was reached, we would raise σ by 20, and we would also readjust $\tau = 0.25/\sigma$. We do not recommend choosing a high σ at all points, or else the algorithm would result in incorrect solutions. If the convergence was not fast enough, after some maximum count, we would switch our convergence criteria to the value function, as the error (between consecutive iterations) seemed to converge to zero. The procedure of raising the σ is reminiscent of CFL conditions for finite-difference schemes, and we are examining how best to choose the PDHG step-sizes σ and τ. The best σ and τ to choose seem to be dependent on the point at which we are computing.

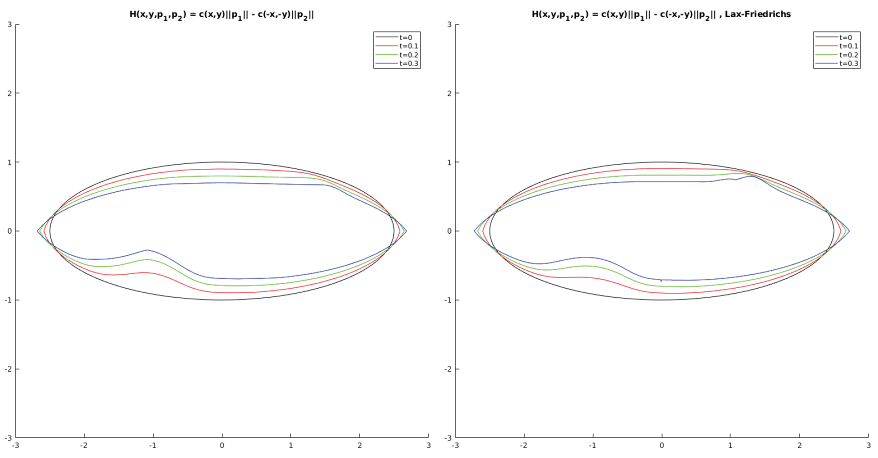

FIG. 6.6. *The difference of norms HJE in two spatial dimensions. This plot shows the zero level sets of the HJE solution for* $t=0.1, 0.2, 0.3$. *We observe that the zero level sets move inward as time increases. Left is computed with our new algorithm, while the right is computed using the conventional Lax-Friedrichs method. Note that there is an anomaly at the top-right of Lax-Friedrich computation. And there is also more of a corner in the bottom-left of the solution computed by the new method. This may be a result of the true solution, and which does not appear in the Lax-Friedrichs solution as it tends to smooth out solutions.*

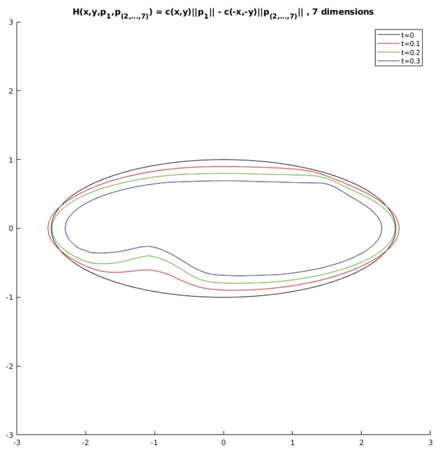

FIG. 6.7. *The difference of norms HJE in seven spatial dimensions. This plot shows the zero level sets of the HJE solution for* $t=0.1, 0.2, 0.3$.

6.3. An (unnamed) example from Isaacs (differential games).

6.3.1. From differential games to HJE for an unnamed example from Isaacs. We now examine a modified version of a differential game, obtained from [15] (Example 6.3), in which it is named "an unnamed example of Isaacs"; the original source is found in Isaac's seminal work [26] (Example 8.4.3). The dynamics are as follows:

$$\begin{cases} \dot{x}(s) = 2\beta + \sin(\alpha) \\ \dot{y}(x) = -c(x,y) + \cos(\alpha) \end{cases}$$

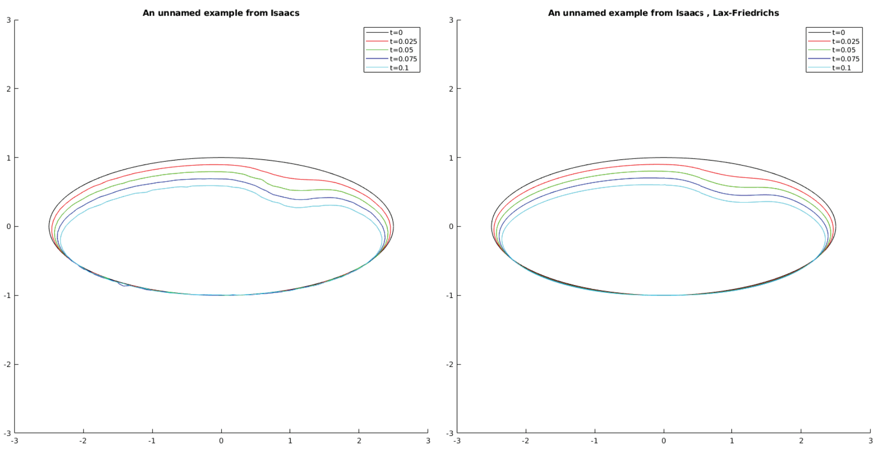

Fig. 6.8. *The zero level-sets for the HJE from an unnamed example of Isaacs. The times we computed were $t=0.025$, 0.05, 0.075, and 0.1. The left figure is the result of our algorithm, while the right figure is the result of Lax-Friedrichs. Here we see that this example is our harshest critic. But this is not surprising because the initial condition is a fully convex function, whereas we'd rather have it to be convex-concave. Also the Hamiltonian is not bounded below with respect to q which is an assumption of the Hopf formula. Neverthless our algorithm is still able to achieve a result similar to Lax-Friedrichs, which might actually be the surprising part. We also note that we only used one initial guess here, but using multiple initial guesses (around 5) smooths out the curves.*

where $0\le\alpha\le 2\pi$ and $-1\le\beta\le 1$. These dynamics are nonlinear. We take the cost-functional as,

$$J[\alpha,\beta]=g(x(0),y(0))+\int_0^t 1\,ds$$

and the value function seeks to maximize with respect to $\alpha\in[0,2\pi]$, and minimize with respect to $\beta\in[-1,1]$. Then our Hamiltonian is,

$$\begin{aligned}H(x,y,p,q)&=\min_{\alpha,\alpha\in[0,2\pi]}\max_{\beta,\beta\in[-1,1]}\left\{\left\langle\begin{pmatrix}2\beta+\sin(\alpha)\\-c(x,y)+\cos(\alpha)\end{pmatrix},\begin{pmatrix}p\\q\end{pmatrix}\right\rangle-1\right\}\\&=\min_{\alpha,\alpha\in[0,2\pi]}\max_{\beta,\beta\in[-1,1]}\{2\beta p-c(x,y)q+p\sin(\alpha)+q\cos(\alpha)-1\}\\&=-c(x,y)q+2|p|-\sqrt{p^2+q^2}-1.\end{aligned}$$

This Hamiltonian is nonconvex and the dynamics are nonlinear.

6.3.2. Implementation details for the (unnamed) example from Isaacs with fully convex initial conditions. We take

$$c(x)=2(1+3\exp(-4\|x-(1,1,0,\dots,0)\|_2^2)),\tag{6.1}$$

which is a positive bump function. The initial condition of our HJE PDE is

$$g(x)=-1/2+(1/2)\langle A^{-1}x,x\rangle,\quad A=\mathrm{diag}(2.5^2,1,0.5^2,\dots,0.5^2).$$

This example is perhaps the harshest on our algorithm and turns out to be slower than coordinate descent, but in many ways this is not surprising. This is because this problem is highly nonconvex and the $g(x,y)$ that we use is a convex function – ideally we would like it to be a convex-concave function which would be suitable for saddle-point problems. Not only that, but our Hamiltonian is not bounded below with respect to q, and the Hopf formula requires this assumption.

Nevertheless, we show this example in order to advertise the generality of our algorithm. It might actually be surprising that our algorithm gives a solution that looks like the Lax-Friedrichs solution. We also note that we only used one initial guess, and in our experiments, using around 5 initial guesses smooths out the solution.

We use Algorithm 3, but modified so we can utilize the convex portion of g (see the last paragraph of Section 5.2). We compute our solutions in a 2-dimensional $[-3,3]^2$ grid. The $\tilde{p}^{k+1}$-update utilizes a combination of gradient descent for the q, and the *shrink*$_2$ operator for the p. The $\tilde{x}^{k+1}$-update uses one step of gradient descent.

We took the time-step as $\delta=0.005$, and we took the PDHG steps as $\sigma=20$, and $\tau=0.25/\sigma$, where the 0.25 comes from the PDHG condition that $\sigma\tau\|D\|_2^2<1$, and $\|D\|_2=2-\varepsilon$ for some small $\varepsilon>0$.

As in the difference of norms example, we did have some points that were slower to converge. We alleviated this problem by rerunning the algorithm at the same point (without changeing σ or τ) if we reached some maximum count, and sometimes took the solution if the maximum count was reached anyway.

The computational time on a $[-3,3]^2$ grid of mesh size $1/12\approx 0.0833$ for each axis, averaged out to 0.412 seconds per point in MATLAB.

In this example, using Algorithm 3 was slower than coordinate descent where it averaged to 0.133 seconds per point, and so we recommend using coordinate descent in this case.

Figure 6.8 gives the result of our algorithm.

6.3.3. Implementation details for the (unnamed) example from Isaacs with convex-concave initial conditions. We take $c(x)$ to be the same as in the fully convex initial conditions (see Equation (6.1)). The initial condition of our HJE PDE is

$$g(x_1,x_2)=-1/2+(1/2)(((2.5)x_1)^2-(x_2)^2).$$

Here we have convex-concave initial conditions, and our algorithm works well. This is an example of a convex-concave game, in which [12] have also applied their method to linear differential games.

We use Algorithm 4, and as in all other examples here, we only have one initial guess. We compute our solutions in a 2-dimensional $[-3,3]^2$ grid. The $\tilde{p}^{k+1}$-update utilizes a combination of gradient descent for the q, and the *shrink*$_2$ operator the p. The $\tilde{x}^{k+1}$-update uses one step of gradient descent.

We took the time-step as $\delta=0.005$, and we took the PDHG steps as $\sigma=2$ for $t=0.025$, 0.05, 0.075, and $\sigma=10$ for $t=0.1$. We always chose $\tau=0.25/\sigma$, where the 0.25 comes from the PDHG condition that $\sigma\tau\|D\|_2^2<1$, and $\|D\|_2=2-\varepsilon$ for some small $\varepsilon>0$.

We computed this example in a $[-3,3]^2$ grid with mesh size $1/12\approx 0.0833$ for each axis. The computational time averaged to 0.125 seconds per point.

As in the difference of norms example, we did have some points that were slower to converge. We alleviated this problem by rerunning the algorithm at the same point (without changing σ or τ) if we reached some maximum count, and sometimes took the solution if the maximum count was reached anyway.

Figure 6.9 gives the result of our algorithm.

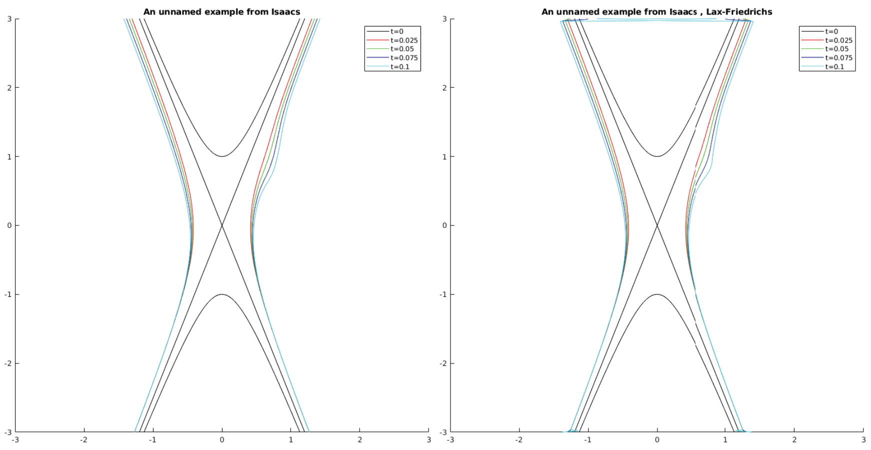

FIG. 6.9. *The zero level-sets for the HJE from an (unnamed) example of Isaacs with convex-concave initial conditions. The times we computed were* $t = 0.025$, 0.05, 0.075, *and* 0.1. *The left figure is the result of our algorithm, while the right figure is the result of Lax-Friedrichs.*

6.4. Quadcopter (a.k.a. Quadrotor or Quad rotorcraft) (optimal control).

6.4.1. From optimal control to HJE for the quadcopter.

A quadcopter is a multirotor helicopter that utilizes four rotors to propel itself across space. The dynamics of a quadcopter [19] are:

$$\begin{cases} \ddot{x} = \frac{u}{m}(\sin(\varphi)\sin(\psi) + \cos(\varphi)\cos(\psi)\sin(\theta)) \\ \ddot{y} = \frac{u}{m}(-\cos(\psi)\sin(\varphi) + \cos(\varphi)\sin(\theta)\sin(\psi)) \\ \ddot{z} = \frac{u}{m}\cos(\theta)\cos(\varphi) - g \\ \ddot{\psi} = \tilde{\tau}_\psi \\ \ddot{\theta} = \tilde{\tau}_\theta \\ \ddot{\varphi} = \tilde{\tau}_\varphi \end{cases}$$

where (x, y, z) is the position of the quadcopter in space, and (ψ, θ, φ) is the angular orientation of the quadcopter (a.k.a. Euler angles). The above second-order system turns into the first-order system,

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{y}_1 = y_2 \\ \dot{z}_1 = z_2 \\ \dot{\psi}_1 = \psi_2 \\ \dot{\theta}_1 = \theta_2 \\ \dot{\varphi}_1 = \varphi_2 \\ \dot{x}_2 = \frac{u}{m}(\sin(\varphi_1)\sin(\psi_1) + \cos(\varphi_1)\cos(\psi_1)\sin(\theta_1)) \\ \dot{y}_2 = \frac{u}{m}(-\cos(\psi_1)\sin(\varphi_1) + \cos(\varphi_1)\sin(\theta_1)\sin(\psi_1)) \\ \dot{z}_2 = \frac{u}{m}\cos(\theta_1)\cos(\varphi_1) - g \\ \dot{\psi}_2 = \tilde{\tau}_\psi \\ \dot{\theta}_2 = \tilde{\tau}_\theta \\ \dot{\varphi}_2 = \tilde{\tau}_\varphi \end{cases}$$

and so the right-side becomes our $\mathbf{f}(\mathbf{x}, \boldsymbol{\alpha})$. Here the controls are the variables u, $\tilde{\tau}_\psi$, $\tilde{\tau}_\theta$, $\tilde{\tau}_\varphi$.

This is a 12-dimensional, nonlinear, optimal control problem.

Denoting $\mathbf{x} = (x_1, y_1, z_1, \psi_1, \theta_1, \varphi_1, x_2, y_2, z_2, \psi_2, \theta_2, \varphi_2)$, then our cost-functional is,

$$J[u, \tilde{\tau}_\psi, \tilde{\tau}_\theta, \tilde{\tau}_\varphi] = g(\mathbf{x}(0)) + \int_0^t 2 + \|(u(s), \tilde{\tau}_\psi(s), \tilde{\tau}_\theta(s), \tilde{\tau}_\varphi(s))\|_2^2 \, ds \tag{6.2}$$

where this cost functional was chosen to follow [42] and [25]. Therefore, our Hamiltonian becomes,

$$\begin{aligned} H(\mathbf{x}, \mathbf{p}, t) = \max_{u, \tilde{\tau}_\psi, \tilde{\tau}_\theta, \tilde{\tau}_\varphi} \Bigg\{ & \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + \begin{bmatrix} \psi_2 \\ \theta_2 \\ \varphi_2 \end{bmatrix} \cdot \begin{bmatrix} p_4 \\ p_5 \\ p_6 \end{bmatrix} \\ & + \frac{u}{m} \begin{bmatrix} \sin(\varphi_1)\sin(\psi_1) + \cos(\varphi_1)\cos(\psi_1)\sin(\theta_1) \\ -\cos(\psi_1)\sin(\varphi_1) + \cos(\varphi_1)\sin(\theta_1)\sin(\psi_1) \\ \cos(\theta_1)\cos(\varphi_1) \end{bmatrix} \cdot \begin{bmatrix} p_7 \\ p_8 \\ p_9 \end{bmatrix} - p_9 g + \begin{bmatrix} \tilde{\tau}_\psi \\ \tilde{\tau}_\theta \\ \tilde{\tau}_\varphi \end{bmatrix} \cdot \begin{bmatrix} p_{10} \\ p_{11} \\ p_{12} \end{bmatrix} \\ & -2 - ||u||^2 - ||\tilde{\tau}_\psi||^2 - ||\tilde{\tau}_\theta||^2 - ||\tilde{\tau}_\varphi||^2 \Bigg\} \end{aligned}$$

$$\begin{aligned} = & \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + \begin{bmatrix} \psi_2 \\ \theta_2 \\ \varphi_2 \end{bmatrix} \cdot \begin{bmatrix} p_4 \\ p_5 \\ p_6 \end{bmatrix} \\ & + \frac{1}{4m} \left\| \begin{bmatrix} \sin(\varphi_1)\sin(\psi_1) + \cos(\varphi_1)\cos(\psi_1)\sin(\theta_1) \\ -\cos(\psi_1)\sin(\varphi_1) + \cos(\varphi_1)\sin(\theta_1)\sin(\psi_1) \\ \cos(\theta_1)\cos(\varphi_1) \end{bmatrix} \cdot \begin{bmatrix} p_7 \\ p_8 \\ p_9 \end{bmatrix} \right\|^2 \\ & - p_9 g + \frac{1}{4}\|p_{10}\|^2 + \frac{1}{4}\|p_{11}\|^2 + \frac{1}{4}\|p_{12}\|^2 - 2. \end{aligned}$$

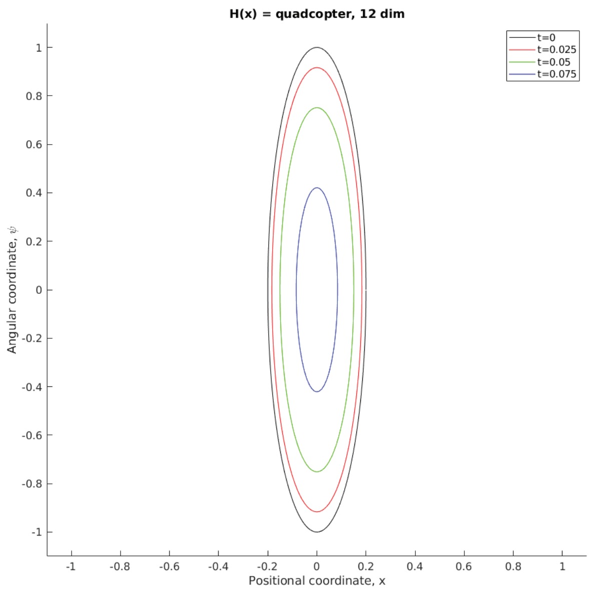

FIG. 6.10. *Here we compute the zero level-sets for the HJE arising from the quadcopter. The x-axis is the x_1-position of the quadcopter, and the y-axis is the angular position in the ψ_1 coordinate. The zero level-sets are computed for $t = 0.025$, 0.05, and 0.075. This is a 12-dimensional, nonlinear optimal control problem.*

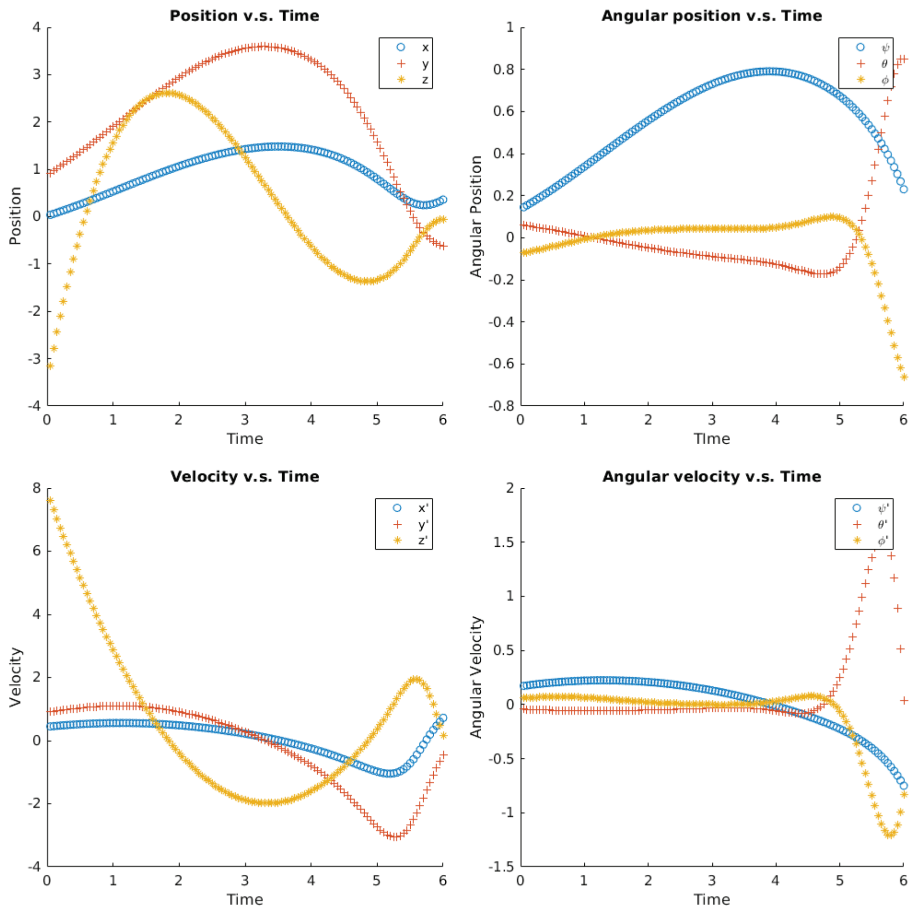

FIG. 6.11. *Here we compute the characteristic curves/optimal trajectories for the quadcopter. We are computing at the terminal point* $\boldsymbol{x}=(0.36,-0.62,-0.06,0.23,0.85,-0.66,0.72,-0.45,0.15,-0.75,0.04,-0.83)$ *and we are computing at the terminal time* $t=6$ *seconds. A plot of the trajectories computed using a different algorithm – SQP – looks identical.*

6.4.2. Implementation details for the quadcopter. Here we have,

$$g(x)=-1/2+(1/2)\langle A^{-1}x,x\rangle, \quad A=\text{diag}(0.2,1,1,\ldots,1).$$

In this case, we use the algorithm based on the generalized Hopf formula, Algorithm 2.

We compute our solutions in a two dimensional slice of $\mathbb{R}^{12}$:

$$[-1,1]\times\{0\}\times\{0\}\times[-1,1]\times\{0\}\times\{0\}\times\{0\}\times\{0\}\times\{0\}\times\{0\}\times\{0\}\times\{0\},$$

i.e. we vary the x-coordinate, as well as the x-velocity-coordinate. Recall that the order of the coordinates are: $\mathbf{x}=(x_1,y_1,z_1,\psi_1,\theta_1,\varphi_1,x_2,y_2,z_2,\psi_2,\theta_2,\varphi_2)$.

For both the $\tilde{p}^{k+1}$-update and the $\tilde{x}^{k+1}$-update, we used gradient descent, except for the update involving $g^*(p_1)$, where we did a proximal-gradient step, i.e. we performed a gradient descent, ignoring g^*, and then we fed the result into $\text{prox}_{\sigma(g^*)}(\cdot)$. This can be seen as a proximal-gradient step [36] (Section 4.2).

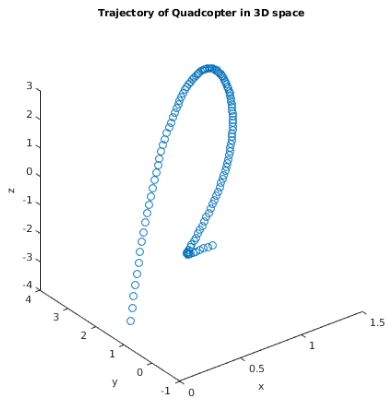

FIG. 6.12. *We plot the (x,y,z) coordinates of the quadcopter to give a plot of the trajectory of the quadcopter in 3D space.*

In this example, we chose as time-step size $\delta = 0.005$, and we chose times $t = 0.025, 0.05, 0.075$. For the PDHG step-sizes, we chose $\sigma = 5$, and $\tau = 0.25/\tau$, where as stated above, the 0.25 comes from the PDHG requirement $\sigma\tau\|D\|_2^2 < 1$, and $\|D\|_2 = 2 - \varepsilon$, for some small $\varepsilon > 0$.

The computation was done on a $[-1,1]^2$ grid, with mesh size 0.01 in each axis. The computational time averaged to 0.0733 seconds per point in MATLAB.

In this case, not only are we able to compute level-sets of the HJE, but we are also able to take advantage of the characteristic curve/optimal trajectory generation that is freely offered by our algorithm.

To generate the curves/trajectories, we took a randomly-generated terminal point, which was exactly

$$x_{\text{target}} = (0.36, -0.62, -0.06, 0.23, 0.85, -0.66, 0.72, -0.45, 0.15, -0.75, 0.04, -0.83)$$

and we computed up to $t = 6$ seconds. We chose a time-step of $\delta = 0.05$, and we chose $\sigma = 11$ (and $\tau = 0.24/\sigma$ (as opposed to 0.25 as the latter will not converge). This took about 24s to compute in MATLAB. We verified our result by directly minimizing a discretized version of (6.2) (see also Section 4.1 on how we discretized), which is a direct collocation method [46]. We performed the optimization using a standard MATLAB minimization solver (fmincon with the SQP algorithm), and this agreed with our results. The solver took 133-347s to compute the trajectories, depending on the accuracy criteria, and we note that fmincon converges to our splitting result the longer we let the algorithm run. So in this case, 5-10+ times speedup is achieved. Computing trajectories at other points have found around an 8-10+ speedups.

Figure 6.10 gives the zero level-sets of the HJE, and fig. 6.11 gives the result of computing the curves/trajectories. Figure 6.12 plots the x, y, and z positions of the quadcopter as it moves through time.

7. Discussion and future work

In this paper, we have presented a splitting method to compute solutions to general (i.e. convex and nonconvex) Hamilton-Jacobi equations which arise from general (i.e. linear and nonlinear) optimal control problems and general differential games problems.

Some nice properties of our algorithm include: (1) relatively fast computation of solutions in high-dimensions, especially when we parallelize the algorithm, which is embarrassingly parallelizable [23] (2) it can generate optimal trajectories of the optimal control/differential games problems, (3) it can compute problems with non-linear ODEs, (4) it can compute solutions for nonconvex Hamiltonians, (5) and the algorithm is embarrassingly parallelizable, i.e. each core can use the algorithm to compute the solution at a point, so given N cores we can compute solutions of the HJE at N points simultaneously.

Splitting applied to optimal control problems has been used by [32] where they apply it to cost functionals having a quadratic and convex term. In terms of Hamilton-Jacobi equations, the authors in Kirchner et al. [30] (2018) effectively applied it to Hamilton-Jacobi equations arising from linear optimal control problems by using the Hopf formula. They make use of the Hopf formula and the closed-form solution to linear ODEs to not only solve HJE, but to also directly compute optimal trajectories in high-dimensional systems. The authors of this current paper have been working in parallel and also applied splitting to HJE and trajectory generation for nonlinear optimal control problems and minimax differential games.

On a related note, see also previous work by Kirchner et al. [29] where they apply the Hopf formula to differential games and show that complex "teaming" behavior can arise, even with linearized pursuit-evasion models.

As far as we know, the idea to use splitting for differential games problem for the discretization in Equations (4.6) and (4.7) is new. We believe it is worth examining if this PDHG-inspired method to solve minimax/saddle-point problems may apply to more general minimax/saddle-point optimization problems.

The proof of convergence and the proof of approximation for our algorithm is still a work-in-progress. But it seems to be that for the examples in Section 6, we get relatively the correct answer, and our algorithm seems to scale linearly with dimension for even a nonlinear optimal control problem requiring nonconvex optimization (see Section 6.1.3).

We also believe that due to the deep connection between Hamilton-Jacobi equations and optimization methods (Section 5.6), it is worthwhile to examine why our algorithm works. Not only that, but for our examples in Section 6, we were able to perform non-convex optimization with only a single initial guess, whereas coordinate descent required multiple. The authors also believe it is worth examining the algorithms – Algorithm 3 and Algorithm 4 – as the computation of minimax differential games problems using a splitting method seems new. In essence, it may be possible to generalize these algorithms to apply to general minimax/saddle-point problems with continuous constraints.

Some improvements to our algorithm for differential games problems (Algorithm 3 and Algorithm 4) can be foreseen:

(1) We have found speedups to our algorithm when we use acceleration methods [3, 4]

(2) One may be able to devise a more sophisticated stopping criteria as that in Kirchner et al. [30], where they apply a step-size-dependent stopping criteria based on the work by [20].

(3) We would also like to utilize higher-order approximations for the ODE and integral when discretizing the value functions of the optimal control or differential games problems. We note that for the Lax discretizations (Algorithm 1 and Algorithm 3), one can average the forward and backward Euler approximations to obtain higher accuracy, analogous to how the trapezoidal approximation is the average of the two.

(4) And we believe we might be able to make use of having a closed-form solution,

or an approximate solution, for computing the characteristic curves, i.e. closed form solutions to $\frac{d}{dt}\mathbf{x}(s)=H_p(\mathbf{x}(s),\mathbf{p}(s),s)$ and $\frac{d}{dt}\mathbf{p}(s)=-H_x(\mathbf{x}(s),\mathbf{p}(s),s)$, much as in [30], where they make use of having a closed-form solution to linear differential equations by utilizing the exponential operator.

(5) There could be an advantage in combining the splitting method with pseudo-spectral methods [41].

(6) In the algorithms for differential equations – Algorithm 3 and Algorithm 4, we are solving a saddle-point problem using gradients. We might obtain faster convergence if we used a Hessian-inspired method, such as split form of BFGS.

(7) Expanding our algorithm so that we may compute with Hamiltonians that do not come in closed-form expressions.

Acknowledgements. We give our deepest thanks to Matthew R. Kirchner and Gary Hewer for their enlightening discussions and suggested edits during this work. They provided us with a wealth of information on the history of the subject and existing methods, as well as important problems. And they were generous in giving suggestions on future directions.

Appendix. A practical tutorial for implementation purposes.

8.1. Optimal control. Suppose we want to compute the Hamilton-Jacobi equations associated to the following optimal control problem:

$$\varphi(x,t)=\min_{\mathbf{x}(\cdot),\mathbf{u}(\cdot)}\left\{g(\mathbf{x}(0))+\int_0^t L(\mathbf{x}(s),\mathbf{u}(s),s)\,ds\right\} \tag{8.1}$$

where $\mathbf{x}(\cdot)$ and $\mathbf{u}(s)$ satisfy the ODE

$$\begin{cases}\dot{\mathbf{x}}(s)=\mathbf{f}(\mathbf{x}(s),\mathbf{u}(s),s), & 0<s<t\\ \mathbf{x}(t)=x.\end{cases} \tag{8.2}$$

Here $(x,t)\in\mathbb{R}^n\times[0,\infty)$ are fixed, and is the point that we want to compute the HJE solution φ.

Then we can use Algorithm 1 or Algorithm 2, which we describe in more detail below.

8.1.1. Practical tutorial for Algorithm 1. If we want to use the discretized Lax formula (with a backward Euler discretization of the ODE dynamics), then:

(1) Discretize the time domain:

$$0=s_0<s_1<s_2<\cdots<s_{N-1}<s_N=t.$$

In our numerical experiments, we chose a uniform discretization of size $\delta:=t/N$.

(2) Approximate (8.2) using backward Euler, and also discretize (8.1) to obtain (as in (4.2)),

$$\varphi(x,t)\approx\max_{\{p_j\}_{j=1}^N}\min_{\{x_j\}_{j=0}^N}\left\{g(x_0)+\sum_{j=1}^N\langle p_j,x_j-x_{j-1}\rangle-\delta\sum_{j=1}^N H(x_j,p_j,s_j)\right\}$$

where $x_j=\mathbf{x}(s_j)$, and $p_j=\mathbf{p}(x_j)$. Let us denote $x=x_{\text{target}}$ to clarify notation.

(3) Initialize:

(a) Choose $\delta > 0$, set $N = t/\delta$ (although note that since we are using the zero-th time-step, we are updating $N+1$ points).
(b) Randomly initialize $\tilde{x}^0 := (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\text{target}}$.
(c) Randomly initialize $\tilde{p}^0 := (p_0^0, p_1^0, \ldots, p_{N-1}^0, p_N^0)$, but with $p_0^0 \equiv 0$, as we won't be updating p_0^0; it is only there for computational accounting.
(d) Set $\tilde{z}^0 := (z_0^0, z_1^0, \ldots, z_N^0) = (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$.
(e) Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0,1]$ (we suggest $\theta = 1$).
(f) Choose some tolerance $\text{tol} > 0$ small.

(4) Set

$$D = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ -I & I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -I & I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \end{pmatrix}$$

where $\mathbf{0}$ is a $(\dim \times \dim)$ zero matrix, where dim is the size of the space variable, and I is the $(\dim \times \dim)$ identity matrix.
Note that in the algorithm, we can replace $(D\tilde{z}^k)_j = z_j^k - z_{j-1}^k$, and similarly with $(D^T\tilde{p}^k)_j = p_j^k - p_{j+1}^k$, so we can save time by not performing a full matrix multiplication.

(5) Then perform the algorithm found in Algorithm 6.

8.1.2. Practical tutorial for Algorithm 2. If we want to use the discretized Hopf formula, then:

(1) Discretize the time domain:

$$0 = s_0 < s_1 < s_2 < \cdots < s_{N-1} < s_N = t.$$

In our numerical experiments, we chose a uniform discretization of size $\delta := t/N$.

(2) Approximate (8.2) using backward Euler, and also discretize (8.1) to obtain (as in (4.4)),

$$\varphi(x,t) \approx \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=1}^N} \left\{ -g^*(p_1) + \langle p_N, x\rangle + \sum_{j=1}^{N-1} \langle p_j - p_{j+1}, x_j \rangle \right. \\ \left. - \delta \sum_{j=1}^{N} H(x_j, p_j, s_j) \right\}$$

where $x_j = \mathbf{x}(s_j)$, and $p_j = \mathbf{p}(x_j)$. Let us denote $x = x_{\text{target}}$ to clarify notation.

(3) Initialize:
(a) Choose $\delta > 0$, set $N = t/\delta$.
(b) Randomly initialize $\tilde{x}^0 := (x_1^0, \ldots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\text{target}}$.
(c) Randomly initialize $\tilde{p}^0 := (p_1^0, \ldots, p_{N-1}^0, p_N^0)$.
(d) Set $\tilde{z}^0 := (z_1^0, \ldots, z_N^0) = (x_1^0, \ldots, x_{N-1}^0, x_N^0)$.

Algorithm 6 Practical tutorial for the Lax formula with backward Euler, for optimal control

```
Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0,\infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 > \text{tol}$ or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 > \text{tol}$) and (count < max_count) do
  for $j = 1$ to $N$ do
    $p_j^{k+1} = \arg\min_p \left\{ \delta H(x_j^k, p, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D\tilde{z}^k)_j)\|_2^2 \right\}$
  end for

  for $j = 0$ do
    $x_0^{k+1} = \arg\min_x \left\{ g(x) + \frac{1}{2\tau}\|x - (x_0^k - \tau(D^T\tilde{p}^k)_0)\|_2^2 \right\}$     (note $p_0^k = 0$)
  end for
  for $j = 1$ to $N-1$ do
    $x_j^{k+1} = \arg\min_x \left\{ -\delta H(x, p_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D^T\tilde{p}^k)_j)\|_2^2 \right\}$
  end for

  for $j = 0$ to $N$ do
    $z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$
  end for
end while

fval $= g(x_0) + \sum_{j=1}^N \langle p_j, x_j - x_{j-1}\rangle - \delta \sum_{j=1}^N H(x_j, p_j, s_j)$
return fval
```

(e) Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0,1]$ (we suggest $\theta = 1$).

(f) Choose some tolerance tol > 0 small.

(4) Set

$$D = \begin{pmatrix} I & -I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I & -I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \end{pmatrix}$$

where $\mathbf{0}$ is a (dim $\times$ dim) zero matrix, where dim is the size of the space variable, and I is the (dim $\times$ dim) identity matrix.

Note we can replace $(D^T\tilde{z^k})_j = z_j^k - z_{j-1}^k$ and $(D\tilde{p}^k)_j = p_j^k - p_{j+1}^k$, so we can save time by not performing a full matrix multiplication.

Also note that the D here is different than in Algorithm 1 and Algorithm 6.

(5) Then perform the algorithm found in Algorithm 7.

8.2. Differential games. Suppose we want to solve the differential games problem with the following dynamics,

$$\begin{cases} \begin{pmatrix} \dot{\mathbf{x}}(s) \\ \dot{\mathbf{y}}(s) \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \\ \mathbf{f}_2(\mathbf{x}(s), \mathbf{y}(s), \boldsymbol{\alpha}(s), \boldsymbol{\beta}(s), s) \end{pmatrix} & 0 < s < t \\ \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \end{cases} \tag{8.3}$$

Algorithm 7 Practical tutorial for the Hopf formula, for optimal control

```
Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0,\infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 > \text{tol}$ or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 > \text{tol}$) and ($\text{count} < \text{max\_count}$) do
  for $j = 1$ do
    $p_j^{k+1} = \arg\min_p \{g^*(p) + \delta H(x_1^k, p, s_1) + \frac{1}{2\sigma}\|p - (p_1^k + \sigma(D^T \tilde{z}^k)_1)\|_2^2\}$
  end for
  for $j = 2$ to $N$ do
    $p_j^{k+1} = \arg\min_p \{\delta H(x_j^k, p, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D^T \tilde{z}^k)_j)\|_2^2\}$
  end for

  for $j = 1$ to $N-1$ do
    $x_j^{k+1} = \arg\min_x \{-\delta H(x, p_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D\tilde{p}^k)_j)\|_2^2\}$
  end for

  for $j = 1$ to $N$ do
    $z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$
  end for
end while

$\text{fval} = -g^*(p_1) + \langle p_N, x_{\text{target}}\rangle + \sum_{j=1}^{N-1}\langle p_j - p_{j+1}, x_j\rangle - \delta\sum_{j=1}^{N} H(x_j, p_j, s_j)$
return fval
```

and with the following value function,

$$\varphi(x,y,t) = \inf_{\boldsymbol{\alpha}(\cdot),\mathbf{x}(\cdot)} \sup_{\boldsymbol{\beta}(\cdot),\mathbf{y}(\cdot)} \left\{ g(\mathbf{x}(0),\mathbf{y}(0)) + \int_0^t L(\mathbf{x}(s),\mathbf{y}(s),\boldsymbol{\alpha}(s),\boldsymbol{\beta}(s),s)\,ds \right\}. \tag{8.4}$$

Then we can discretize the above equation and use Algorithm 3 and Algorithm 4, which we describe in more detail below.

8.2.1. Practical tutorial for Algorithm 3. If we want to use the discretized Lax formula for differential games (with a backward Euler discretization of the ODE dynamics) (4.6), then:

(1) Discretize the time domain:

$$0 = s_0 < s_1 < s_2 < \cdots < s_{N-1} < s_N = t.$$

In our numerical experiments, we chose a uniform discretization of size $\delta := t/N$.

(2) Approximate (8.3) using backward Euler, and also discretize (8.4) to obtain (as in (4.6)),

$$\varphi(x,y,t) \approx \min_{\{q_j\}_{j=1}^N} \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=0}^N} \max_{\{y_j\}_{j=0}^N} \left\{ g(x_0,y_0) + \sum_{j=1}^N \left\langle \begin{pmatrix} p_j \\ -q_j \end{pmatrix}, \begin{pmatrix} x_j - x_{j-1} \\ y_j - y_{j-1} \end{pmatrix} \right\rangle \right.$$
$$\left. - \delta \sum_{j=1}^N H(x_j,y_j,p_j,-q_j,s_j) \right\}$$

Algorithm 8 Practical tutorial for the Lax formula with backward Euler, for differential games

```
Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0,\infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 > \text{tol}$ or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 > \text{tol}$ or $\|\tilde{y}^{k+1} - \tilde{y}^k\|_2^2 > \text{tol}$ or $\|\tilde{q}^{k+1} - \tilde{q}^k\|_2^2 > \text{tol}$) and (count $<$ max_count) do
    for $j = 1$ to $N$ do
        $p_j^{k+1} = \arg\min_p \left\{\delta H(x_j^k, y_j^k, p, -q_j^k, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D\tilde{z}^k)_j)\|_2^2\right\}$
    end for

    for $j = 1$ to $N$ do
        $q_j^{k+1} = \arg\min_q \left\{-\delta H(x_j^k, y_j^k, p_j^{k+1}, -q, s_j) + \frac{1}{2\sigma}\|q - (q_j^k + \sigma(D\tilde{w}^k)_j)\|_2^2\right\}$
    end for

    for $j = 0$ do
        $x_0^{k+1} = \arg\min_x \left\{g(x, y_0^k) + \frac{1}{2\tau}\|x - (x_0^k - \tau(D^T\tilde{p}^k)_0)\|_2^2\right\}$
    end for
    for $j = 1$ to $N-1$ do
        $x_j^{k+1} = \arg\min_x \left\{-\delta H(x, y_j^k, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D^T\tilde{p}^k)_j)\|_2^2\right\}$
    end for

    for $j = 0$ do
        $y_0^{k+1} = \arg\min_y \left\{-g(x_0^{k+1}, y) + \frac{1}{2\tau}\|y - (y_0^k - \tau(D^T\tilde{q}^k)_0)\|_2^2\right\}$
    end for
    for $j = 1$ to $N-1$ do
        $y_j^{k+1} = \arg\min_y \left\{-\delta H(x_j^{k+1}, y, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|y - (y_j^k - \tau(D^T\tilde{q}^k)_j)\|_2^2\right\}$
    end for

    for $j = 0$ to $N$ do
        $z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$
        $w_j^{k+1} = y_j^{k+1} + \theta(y_j^{k+1} - y_j^k)$
    end for
end while

fval $= g(x_0, y_0) + \sum_{j=1}^N \left\langle \begin{pmatrix} p_j \\ -q_j \end{pmatrix}, \begin{pmatrix} x_j - x_{j-1} \\ y_j - y_{j-1} \end{pmatrix} \right\rangle - \delta \sum_{j=1}^N H(x_j, y_j, p_j, -q_j, s_j)$
return fval
```

where $x_j = \mathbf{x}(s_j)$, and similarly for y_j, q_j, and p_j. Let us denote $x = x_{\text{target}}$ and $y = y_{\text{target}}$ to clarify notation.

(3) Initialize:

1) Choose $\delta > 0$, set $N = t/\delta$ (although note that since we are using the zero-th time-step, we are updating $N+1$ points).
2) Randomly initialize $\tilde{x}^0 := (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\text{target}}$, and similarly for $\tilde{y}^0$.
3) Randomly initialize $\tilde{p}^0 := (p_0^0, p_1^0, \ldots, p_{N-1}^0, p_N^0)$, but with $p_0^0 \equiv 0$, as we won't be

updating p_0^0; it is only there for computational accounting. Do a similar initialization for $\tilde{q}^0$.

4) Set $\tilde{z}^0 := (z_0^0, z_1^0, \dots, z_N^0) = (x_0^0, x_1^0, \dots, x_{N-1}^0, x_N^0)$, and set $\tilde{w}^0 := (w_0^0, w_1^0, \dots, w_N^0) = (y_0^0, y_1^0, \dots, y_{N-1}^0, y_N^0)$.
5) Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0,1]$ (we suggest $\theta = 1$).
6) Choose some tolerance $\mathrm{tol} > 0$ small.

(4) Set

$$D = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ -I & I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -I & I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \end{pmatrix}$$

where $\mathbf{0}$ is a $(\dim \times \dim)$ zero matrix, where dim is the size of the space variable, and I is the $(\dim \times \dim)$ identity matrix. Note that this is a very sparse matrix and we take advantage of this.

Note that we can replace $D\tilde{z}^k = z_j^k - z_{j-1}^k$, and $D\tilde{w}^k = w_j^k - w_{j-1}^k$, and $D^T\tilde{p}^k = p_j^k - p_{j+1}^k$, and $D^T\tilde{q}^k = q_j^k - q_{j+1}^k$, so we don't have to perform the full matrix multiplication.

(5) Then perform the algorithm found in Algorithm 8.

8.2.2. Practical tutorial for Algorithm 4. If we want to use the discretized Hopf formula for differential games (with a backward Euler discretization of the ODE dynamics) (4.7), then:

(1) Discretize the time domain:

$$0 = s_0 < s_1 < s_2 < \cdots < s_{N-1} < s_N = t.$$

In our numerical experiments, we chose a uniform discretization of size $\delta := t/N$.

(2) Approximate (8.3) using backward Euler, and also discretize (8.4) to obtain (as in (4.7)),

$$\varphi(x,y,t) \approx \min_{\{q_j\}_{j=1}^N} \max_{\{p_j\}_{j=1}^N} \min_{\{x_j\}_{j=1}^N} \max_{\{y_j\}_{j=1}^N} \Bigg\{ -e^*(p_1) - h_*(-q_1) + \left\langle \begin{pmatrix} p_N \\ -q_N \end{pmatrix}, \begin{pmatrix} x \\ y \end{pmatrix} \right\rangle + \sum_{j=1}^{N-1} \left\langle \begin{pmatrix} p_j - p_{j+1} \\ -(q_j - q_{j+1}) \end{pmatrix}, \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\rangle - \delta \sum_{j=1}^{N} H(x_j, y_j, p_j, -q_j, s_j) \Bigg\}$$

where $x_j = \mathbf{x}(s_j)$, and similarly for y_j, q_j, and p_j. Let us denote $x = x_{\mathrm{target}}$ and $y = y_{\mathrm{target}}$ to clarify notation.

(3) Initialize:
1) Choose $\delta > 0$, set $N = t/\delta$.
2) Randomly initialize $\tilde{x}^0 := (x_0^0, x_1^0, \dots, x_{N-1}^0, x_N^0)$, but with $x_N^0 \equiv x_{\mathrm{target}}$, and similarly for $\tilde{y}^0$.
3) Randomly initialize $\tilde{p}^0 := (p_0^0, p_1^0, \dots, p_{N-1}^0, p_N^0)$, but with $p_0^0 \equiv 0$, as we won't be updating p_0^0; it is only there for computational accounting. Do the same initialization for $\tilde{q}^0$.

Algorithm 9 Practical tutorial for the Hopf formula with backward Euler, for differential games

```
Given: $x_{\text{target}} \in \mathbb{R}^d$ and time $t \in (0,\infty)$.

while ($\|\tilde{x}^{k+1} - \tilde{x}^k\|_2^2 > \text{tol}$ or $\|\tilde{p}^{k+1} - \tilde{p}^k\|_2^2 > \text{tol}$ or $\|\tilde{y}^{k+1} - \tilde{y}^k\|_2^2 > \text{tol}$ or $\|\tilde{q}^{k+1} - \tilde{q}^k\|_2^2 > \text{tol}$) and (count $<$ max_count) do
  for $j=1$ do
    $p_1^{k+1} = \arg\min_p \left\{ e^*(p) + \delta H(x_1^k, y_1^k, p, -q_1^k, s_1) + \frac{1}{2\sigma}\|p - (p_1^k + \sigma(D^T\tilde{z}^k)_1)\|_2^2 \right\}$
  end for
  for $j=2$ to $N$ do
    $p_j^{k+1} = \arg\min_p \left\{ \delta H(x_j^k, y_j^k, p, -q_j^k, s_j) + \frac{1}{2\sigma}\|p - (p_j^k + \sigma(D^T\tilde{z}^k)_j)\|_2^2 \right\}$
  end for

  for $j=1$ do
    $q_1^{k+1} = \arg\min_q \left\{ -h_*(-q) - \delta H(x_1^k, y_1^k, p_1^{k+1}, -q, s_1) + \frac{1}{2\sigma}\|q - (q_1^k + \sigma(D^T\tilde{w}^k)_1)\|_2^2 \right\}$
  end for
  for $j=2$ to $N$ do
    $q_j^{k+1} = \arg\min_q \left\{ -\delta H(x_j^k, y_j^k, p_j^{k+1}, -q, s_j) + \frac{1}{2\sigma}\|q - (q_j^k + \sigma(D^T\tilde{w}^k)_j)\|_2^2 \right\}$
  end for

  for $j=1$ to $N-1$ do
    $x_j^{k+1} = \arg\min_x \left\{ -\delta H(x, y_j^k, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|x - (x_j^k - \tau(D\tilde{p}^k)_j)\|_2^2 \right\}$
  end for

  for $j=1$ to $N-1$ do
    $y_j^{k+1} = \arg\min_y \left\{ -\delta H(x_j^{k+1}, y, p_j^{k+1}, -q_j^{k+1}, s_j) + \frac{1}{2\tau}\|y - (y_j^k - \tau(D\tilde{q}^k)_j)\|_2^2 \right\}$
  end for

  for $j=1$ to $N$ do
    $z_j^{k+1} = x_j^{k+1} + \theta(x_j^{k+1} - x_j^k)$
    $w_j^{k+1} = y_j^{k+1} + \theta(y_j^{k+1} - y_j^k)$
  end for
end while

$\text{fval} = -e^*(p_1) - h_*(-q_1) + \left\langle \begin{pmatrix} p_N \\ -q_N \end{pmatrix}, \begin{pmatrix} x_{\text{target}} \\ y_{\text{target}} \end{pmatrix} \right\rangle + \sum_{j=1}^{N-1} \left\langle \begin{pmatrix} p_j - p_{j+1} \\ -(q_j - q_{j+1}) \end{pmatrix}, \begin{pmatrix} x_j \\ y_j \end{pmatrix} \right\rangle - \delta \sum_{j=1}^{N} H(x_j, y_j, p_j, -q_j, s_j)$
return fval
```

4) Set $\tilde{z}^0 := (z_0^0, z_1^0, \ldots, z_N^0) = (x_0^0, x_1^0, \ldots, x_{N-1}^0, x_N^0)$, and set $\tilde{w}^0 := (w_0^0, w_1^0, \ldots, w_N^0) = (y_0^0, y_1^0, \ldots, y_{N-1}^0, y_N^0)$.

5) Choose σ, τ such that $\sigma\tau < 1/\|D\|_2^2 = 0.25$ and $\theta \in [0,1]$ (we suggest $\theta = 1$).

6) Choose some tolerance $\text{tol} > 0$ small.

(4) Set

$$D = \begin{pmatrix} I & -I & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I & -I & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & I \end{pmatrix}$$

where $\mathbf{0}$ is a $(\dim \times \dim)$ zero matrix, where dim is the size of the space variable, and I is the $(\dim \times \dim)$ identity matrix. Note that this is a very sparse matrix and we take advantage of this.

Also note that we can replace $D^T \tilde{z}^k = z_j^k - z_{j-1}^k$, and $D^T \tilde{w}^k = w_j^k - w_{j-1}^k$, and $D\tilde{p}^k = p_j^k - p_{j+1}^k$, and $D\tilde{q}^k = q_j^k - q_{j+1}^k$, so we don't have to perform the full matrix multiplication.

(5) Then perform the algorithm found in Algorithm 9.

REFERENCES

[1] S. Bansal, M. Chen, S. L. Herbert, and C. J. Tomlin, *Hamilton-Jacobi reachability: A brief overview and recent advances*, in Proc. of 56th IEEE Conf. Decis. Control, 2017.

[2] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Foundations and Trends in Machine Learning, 3(1):1–122, 2011.

[3] A. Chambolle and T. Pock, *A first-order primal-dual algorithm for convex problems with applications to imaging*, J. Math. Imaging Vision, 40(1):120–145, 2011.

[4] A. Chambolle and T. Pock, *On the ergodic convergence rates of a first-order primal–dual algorithm*, Math. Program., 159(1):253–287, 2016.

[5] P. Chaudhari, A. M. Oberman, S. Osher, S. Soatto, and G. Carlier, *Deep relaxation: partial differential equations for optimizing deep neural networks*, Res. Math. Sci., 5: 30, 2018.

[6] Y. T. Chow, J. Darbon, S. Osher, and W. Yin, *Algorithm for overcoming the curse of dimensionality for certain non-convex Hamilton-Jacobi equations, projections and differential games*, Ann. Math. Sci. Appl., 3(2):369–403, 2018.

[7] Y. T. Chow, J. Darbon, S. Osher, and W. Yin, *Algorithm for overcoming the curse of dimensionality for time-dependent non-convex Hamilton–Jacobi equations arising from optimal control and differential games problems*, J. Sci. Comput., 73(2):617–643, 2017.

[8] Y. T. Chow, J. Darbon, S. Osher, and W. Yin, *Algorithm for overcoming the curse of dimensionality for state-dependent Hamilton-Jacobi equations*, UCLA CAM Report 17-16, 2017.

[9] J. Darbon, *On convex finite-dimensional variational methods in imaging sciences and Hamilton–Jacobi equations*, SIAM J. Imag. Sci., 8(4):2268–2293, 2015.

[10] J. Darbon and S. Osher, *Algorithms for overcoming the curse of dimensionality for certain Hamilton–Jacobi equations arising in control theory and elsewhere*, Res. Math. Sci., 3(1):19, 2016.

[11] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1(1):269–271, 1959.

[12] P. Dvurechensky, Y. Nesterov, and V. Spokoiny, *Primal-dual methods for solving infinite-dimensional games*, J. Opt. Theo. Appl., 166(1):23–51, 2015.

[13] R. J. Elliott and N. J. Kalton, *Values in differential games*, Bull. Amer. Math. Soc., 78(3):427–431, 1972.

[14] E. Esser, X. Zhang, and T. F. Chan, *A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science*, SIAM J. Imaging Sci., 3(4):1015–1046, 2010.

[15] L. C. Evans, *Envelopes and nonconvex Hamilton–Jacobi equations*, https://www.docin.com/p-948311797.html.

[16] L. C. Evans, *Partial Differential Equations*, Amer. Math. Soc., Providence, R.I., 2010.

[17] L. C. Evans, *An introduction to mathematical optimal control theory version 0.2*, 2017.

[18] L. C. Evans and P. E. Souganidis, *Differential games and representation formulas for solutions of Hamilton-Jacobi-Isaacs equations*, Indiana Univ. Math. J., 33(5):773–797, 1984.

[19] L. R. G. Carrillo, A. E. D. López, R. Lozano, and C. Pégard, *Modeling the quad-rotor minirotorcraft*, in Quad Rotorcraft Control, Springer London, London, 23–34, 2013.

[20] T. Goldstein, M. Li, and X. Yuan, *Adaptive primal-dual splitting methods for statistical learning and image processing*, in Advances in Neural Information Processing Systems, 2089–2097, 2015.

[21] T. Goldstein and S. Osher, *The split Bregman method for L1-regularized problems*, SIAM J. Imag. Sci., 2(2):323–343, 2009.

[22] R. F. Hartl, S. P. Sethi, and R. G. Vickson, *A survey of the maximum principles for optimal control problems with state constraints*, SIAM Rev., 37(2):181–218, 1995.

[23] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[24] E. Hopf, *Generalized solutions of non-linear equations of first order*, J. Math. Mech., 14(6):951–973, 1965.

[25] M. B. Horowitz, A. Damle, and J. W. Burdick, *Linear Hamilton Jacobi Bellman equations in high dimensions*, in Proc. of 53rd IEEE Conf. Decis. Control, 5880–5887, 2014.

[26] R. P. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*, Dover Publications, 1999.

[27] D. Kalise and K. Kunisch, *Polynomial approximation of high-dimensional Hamilton-Jacobi-Bellman equations and applications to feedback control of semilinear parabolic PDEs*, ArXiv e-prints, 2017.

[28] W. Kang and L. C. Wilcox, *Mitigating the curse of dimensionality: Sparse grid characteristics method for optimal feedback control and HJB equations*, Comput. Opt. Appl., 68(2):289–315, 2017.

[29] M. R. Kirchner, R. Mar, G. Hewer, J. Darbon, S. Osher, and Y.T. Chow, *Time-optimal collaborative guidance using the generalized Hopf formula*, IEEE Control Syst. Lett., 2(2):201–206, 2018.

[30] M. Kirchner, G. Hewer, J. Darbon, and S. Osher, *A primal-dual method for optimal control and trajectory generation in high-dimensional systems*, UCLA CAM Report 18-04, 2018.

[31] I. M. Mitchell, M. Chen, and M. Oishi, *Ensuring safety of nonlinear sampled data systems through reachability*, IFAC Proceedings Volumes (Special Issue for 4th IFAC Conference on Analysis and Design of Hybrid Systems), 45(9):108–114, 2012.

[32] B. O'Donoghue, G. Stathopoulos, and S. Boyd, *A splitting method for optimal control*, IEEE Trans. Control Syst. Technol., 21(6):2432–2442, 2013.

[33] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer–Verlag New York, Inc., 2003.

[34] S. Osher and J. A. Sethian, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations*, J. Comput. Phys., 79(1):12–49, 1988.

[35] S. Osher and C.-W. Shu, *High-order essentially nonoscillatory schemes for Hamilton–Jacobi equations*, SIAM J. Numer. Anal., 28(4):907–922, 1991.

[36] N. Parikh and S.Boyd, *Proximal algorithms*, Foundations and Trends in Optimization, 1(3):127–239, 2014.

[37] L. Qi and W. Sun, *An iterative method for the minimax problem*, in D.-Z. DuPanos and P. M. Pardalos (Eds.), Minimax and Applications, Springer US, Boston, MA, 55–67, 1995.

[38] R. Rockafellar, *Conjugate Duality and Optimization*, Soc. Indust. Appl. Math., 1974.

[39] R. T. Rockafellar and P. R. Wolenski, *Convexity in Hamilton–Jacobi theory I: Dynamics and duality*, SIAM J. Control Optim., 39(5):1323–1350, 2000.

[40] R. T. Rockafellar and P. R. Wolenski, *Convexity in Hamilton–Jacobi theory II: Envelope representations*, SIAM J. Control Optim., 39(5):1351–1372, 2000.

[41] I. M. Ross and F. Fahroo, *Legendre pseudospectral approximations of optimal control problems*, in W. Kang, C. Borges, and M. Xiao (Eds.), New Trends in Nonlinear Dynamics and Control and their Applications, Springer Berlin Heidelberg, 327–342, 2003.

[42] E. Stefansson and Y.P. Leong, *Sequential alternating least squares for solving high dimensional linear Hamilton-Jacobi-Bellman equation*, in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 3757–3764, 2016.

[43] Y.-H. R. Tsai, L.-T. Cheng, S. Osher, and H.-K. Zhao, *Fast sweeping algorithms for a class of Hamilton–Jacobi equations*, SIAM J. Numer. Anal., 41(2):673–694, 2003.

[44] J. N. Tsitsiklis, *Efficient algorithms for globally optimal trajectories*, in Proc. of 33rd IEEE Conf. Decis. Control, 2:1368–1373, 1994.

[45] P.P. Varaiya, *Differential games*, in Probability Theory (Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability), University of California Press, Berkeley 3:687–697, 1972.

[46] O. von Stryk, *Numerical solution of optimal control problems by direct collocation*, in R. Bulirsch,

A. Miele, J. Stoer, and K. Well (Eds.), Optimal Control, Birkhäuser Basel, Basel, 129–143, 1993.

[47] J. Yong, *Differential Games: A Concise Introduction*, World Scientific, 2014.

[48] M. Zhu and T. Chan, *An efficient primal-dual hybrid gradient algorithm for total variation image restoration*, UCLA CAM Report, 2008.