LevelSetPy: A GPU-Accelerated Package for Resolving Hyperbolic Hamilton-Jacobi Partial Differential Equations

LEKAN MOLU, Microsoft Research, USA

This technical communiqué is devoted to several themes surrounding the numerical solution to hyperbolic Hamilton-Jacobi (HJ) partial differential equations (PDEs) *viz*., their implicit representation (as interfaces) on co-dimension one domains; dynamics evolution with levelsets; upwinding spatial derivatives; total variation diminishing Runge-Kutta integration schemes via method of lines; and their applications to the theory of reachable sets and safety-critical systems analysis. The Cauchy-type HJ evolution equation and its level set *dynamics* that we chiefly consider[1] are increasingly finding interest in multiple research domains: (a) they are useful for analyzing safety-critical problems in reinforcement learning, and broadly in robotics and automation; (b) control engineering; (c) computing flows in transport problems; (d) aerospace and defense engineering domains; (e) geometric optics; (f) volume of fluid analysis; and (g) image processing. We briefly review HJ theory and its viscosity approximations, describe a hierarchy of library components, and provide rigorously-tested representative numerical examples in reachability differential games, transport analysis, and time-optimal control problems. This software package allows for easy portability and extensibility to modern libraries for the analyses of safety-critical algorithms in (reinforcement) learning, control, robotics, transport, and flow problems *inter alia*. Let us enquire.

CCS Concepts: • **Software and its engineering → Software libraries and repositories**; • **Applied computing** → *Physical sciences and engineering*; • **Mathematics of computing** → *Solvers*.

Additional Key Words and Phrases: partial differential equations, level sets, reachability theory

ACM Reference Format:
Lekan Molu. 20XX. LevelSetPy: A GPU-Accelerated Package for Resolving Hyperbolic Hamilton-Jacobi Partial Differential Equations. *J. ACM* 00, 0, Article 000 (20XX), 26 pages. https://doi.org/XXXXXXX.XXXXXXX

1 OVERVIEW

The need for scalable and faster numerical algorithms in software for *verifying* and *validating* systems has become timely given the emerging growth of complexity in the systems that we design and build. That which entails generating evidence that a system or one or more of its components satisfy all specified requirements and functional and allocated baselines is termed verification [9]. Whereas, validation entails providing unreprovable evidence that system capabilities comply with an end-user's performance requirements and satisfies its intended operational environment [8]. We focus on verification in this article. This is commonly referred to as safety (ascertainig the freedom of a system from harm) in software design and programming language, aerospace and defense, as well as control theory literature. For the rest of this article safety shall be taken to mean verification. The foremost open-source verification software for engineering applications based on Hamilton-Jacobi

[1]The package is available on the author's github repository: https://github.com/robotsorcerer/LevelSetPy.

Author's address: Lekan Molu, lekanmolu@microsoft.com, Microsoft Research, 300 Lafayette Street, New York, NY, USA, 10012.

0004-5411/20XX/0-ART000 $15.00
https://doi.org/XXXXXXX.XXXXXXX

equations [13, 17] and level set methods [26, 31] is the CPU-based MATLAB® [22] level sets toolbox, written in 2005 A.D. before the era of graphical processing units (GPU)-accelerated computations. Since, there has been a lot of improvement in computing hardware design, architecture, and code-acceleration. This paper principally describes a GPU-accelerated software package – written entirely in python and accelerated on graphical processing units (GPUs) with CuPy [25] – for numerically resolving generalized discontinuous solutions to Cauchy-type (or time-dependent) Hamilton-Jacobi (HJ) hyperbolic partial differential equations (PDE's) that arise in many problem contexts including (multi-agent) reinforcement learning, robotics, control theory, differential games, as well as flow and transport phenomena *inter alia*.

Accompanying the package are implicit calculus operations on dynamic codimension-one implicit interfaces embedded on surfaces in $\mathbb{R}^n$, and numerical (spatial and temporal) discretization schemes for hyperbolic partial differential equations. Furthermore, we describe explicit integration schemes including Lax-Friedrichs, Courant-Friedrichs-Lewy (CFL) integration conditioning for TVD-RK schemes for HJ Hamiltonians of the form $\boldsymbol{H}(\boldsymbol{x}, \boldsymbol{p})$, where $\boldsymbol{x}$ is the state and $\boldsymbol{p}$ is the co-state variable. Finally, extensions to reachability analysis for continuous and hybrid systems, formulated as optimal control or game theory problems using viscosity solutions to HJ problems is described. While our emphasis is on the resolution of safe sets in a reachability context for verification settings, the applications of the software package herewith presented extend beyond control engineering applications.

2 BACKGROUND AND MOTIVATION

Our chief interest is the evolution form of the HJ equation

$$\begin{aligned} \boldsymbol{v}_t(x,t) + \boldsymbol{H}(t; x, \nabla_x \boldsymbol{v}) = 0 \text{ in } \Omega \times (0,T] \\ \boldsymbol{v}(x,t) = \boldsymbol{g}, \text{ on } \partial\Omega \times \{t = T\}, \boldsymbol{v}(x,0) = \boldsymbol{v}_0(x) \text{ in } \Omega \end{aligned} \tag{1}$$

or the convection equation

$$\begin{aligned} \boldsymbol{v}_t + \sum_{i=0}^{N} f_i(u)_{x_i} = 0, \quad \text{for } t > 0, \boldsymbol{x} \in \mathbb{R}^n, \\ \boldsymbol{v}(\boldsymbol{x},0) = \boldsymbol{v}_0(\boldsymbol{x}), \ \boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{2}$$

where Ω is an open set in $\mathbb{R}^n$; $\boldsymbol{x}$ is the state; $\boldsymbol{v}_t$ denotes the partial derivative(s) of the solution $\boldsymbol{v}$ with respect to time t; the Hamiltonian $\boldsymbol{H} : (0,T] \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ and f are continuous; $\boldsymbol{g}$, and $\boldsymbol{v}_0$ are bounded and uniformly continuous (BUC) functions in $\mathbb{R}^n$; and $\nabla_x \boldsymbol{v}$ is the spatial gradient of $\boldsymbol{v}$. It is assumed that $\boldsymbol{g}$ and $\boldsymbol{v}_0$ are given.

Solving problems described by (1) under appropriate boundary and/or initial conditions using the method of characteristics is limiting as a result of crossing characteristics [5]. In the same vein, global analysis is virtually impossible owing to the lack of existence and uniqueness of solutions $\boldsymbol{v} \in C^1(\Omega) \times (0,T]$ even if $\boldsymbol{H}$ and $\boldsymbol{g}$ are smooth [5].

The method of "vanishing viscosity", based on the idea of traversing the limit as $\delta \to 0$ in the hyperbolic equation (1) (where a parameter $\delta > 0$ "endows" the problem in a viscosity sense as in gas dynamics [14]), allows generalized (discontinuous) solutions [11] whereupon if $\boldsymbol{v} \in W^{1,\infty}_{loc}(\Omega) \times (0,T]$ and $\boldsymbol{H} \in W^{1,\infty}_{loc}(\Omega)$, one can lay claim to strong notions of general existence, stability, and uniqueness to BUC solutions $\boldsymbol{v}^\delta$ of the (approximate) viscous Cauchy-type HJ equation

$$\begin{aligned} \boldsymbol{v}^\delta_t + \boldsymbol{H}(t; \boldsymbol{x}, \nabla_x \boldsymbol{v}^\delta) - \delta \Delta \boldsymbol{v}^\delta = 0 \text{ in } \Omega \times (0,T] \\ \boldsymbol{v}^\delta(x,t) = \boldsymbol{g}, \text{ on } \partial\Omega \times \{t = T\}, \boldsymbol{v}^\delta(\boldsymbol{x},0) = \boldsymbol{v}_0(\boldsymbol{x}) \text{ in } \Omega \end{aligned} \tag{3}$$

in the class $\mathrm{BUC}(\Omega \times [0,T]) \cap C^{2,1}(\Omega \times (0,T])$ i.e. continuous second-order spatial and first order time derivatives for all time $T < \infty$. Crandall and Lions [6] showed that $|v^{\delta}(x,t) - v(x,t)| \leq k\sqrt{\delta}$ for $x \in \Omega$ and $t > 0$. For most of this article, we are concerned with *generalized* viscosity solutions of the manner described by (3).

Popular means of computing the solution v^{δ} of (3) leverage hyperbolic conservation laws in generating *propagation of surfaces under curvature* (PSC) [27]. These PSC algorithms *implicitly* embed $(n-1)$ dimensional *interfaces* (or isocontours) between two or more separable regions in spatial dimensions belonging in $\mathbb{R}^n$. The interface's speed is typically set as a function of the dynamical system's curvature, and the front gets passively advected by a coupled flow.

This implicit representation has found solution in a wide array of flow/motion of many physical phenomena such as gas dynamics, global illumination problems, and problems arising in the evolution of the trajectories of dynamical systems in a reachable control context [21, 23, 24]. These schemes work despite these phenomena developing discontinuities, corners, or cusps as dynamics evolve owing to the rich theory of viscosity solutions to HJ problems [4, 5, 11, 18].

Mitchell [24] connected techniques used in level set methods to reachability analysis in optimal control, essentially showing that the zero-level set of the differential zero-sum two-person game in an Hamilton-Jacobi-Isaacs (HJI) setting [11, 15] constitutes the safe set of a reachability problem. Reachability concerns evaluating the *decidability* of a dynamical system's trajectories' evolution throughout a state space. Decidable reachable systems are those where one can compute all states that can be reached from an initial condition in *a finite number of steps*. For inf-sup or sup-inf optimal control problems [19], the Hamiltonian is related to the *backward* reachable set of a dynamical system. The essential fabric of this techical communique revolves around reachability problems defined on co-dimension-one surfaces.

The well-known `LevelSet Toolbox` [22] is the consolidated `MATLAB®` package that contains the gridding methods, boundary conditions, time and spatial derivatives, integrators and helper functions. While Mitchell motivated the execution of the toolkit in `MATLAB®` for the expressiveness the language provides, modern data manipulation and scripting libraries often render the original package non-portable across distributed hardware since it lacks other programming language interoperability, particularly python and its associated scientific computing libraries such as `Numpy`, `Scipy`, `PyTorch` and their variants.

In this regard, we revisit the major algorithms necessary for implicit surface representation of HJ PDEs, write the spatial, temporal, and monotone difference schemes in Python, accelerated onto GPUs via CuPy [25] and present representative numerical examples.

2.1 Contributions

Given the prevalence of reachability analysis in application areas encompassing robotics and learning-enabled systems, the need for easy portability and extensibility to other popular software packages has necessitated our rewriting the toolbox in a single python package, leveraging GPU computations, and covering the essential algorithms of Crandall [7], Osher and Fedkiw [26], and Mitchell [22]. Our contributions are as follows:

(1) we describe the details of the LevelSetPy software, starting with the common implicit surfaces that are used as initial conditions to represent $v(x,t)$ leveraging signed distance functions (SDF) and the associated boolean operations and calculus toolboxes for their *dynamics* endowment;
(2) we describe the upwinding spatial derivative schemes, temporal discretization via method of lines schemes based on (approximate) total variation diminishing (TVD) Runge-Kutta (RK), and stabilizing Lax-Friedrichs schemes for multidimensional monotone Hamiltonians of HJ equations or scalar conservation laws;

(3) we conclude this article with three representative examples, viz., (i) the barrier surface for two adversarial rockets traveling on a plane [10]; (ii) the time-to-reach time-optimal control problem for a generic double integrator system; and (iii) analyze the barrier surface for a flock of birds traversing a phase space [20].

To remain as close in spirit as possible to the original MATLAB levelset toolbox [22], we have retained the documentation for the original codes in our implementations.

This article provides a description of the underlying theory and implementation of these hyperbolic PDE's. First, in section 3, differential games in the context of dynamic programming and reachability theory are introduced. Second, in section 4, we describe the geometry of (and Boolean operations on) implicit function representations of continuous-time value functions described by (1) using Cartesian grids. Third, spatial derivatives to scalar conservation laws are described in 5; and temporal discretization schemes for these conservation laws follow. Fourth, we describe real-world problems and make them amenable to HJ PDE's and the construction of safe sets or tubes within a verification geometrical reasoning framework. Fifth, we present numerical results on the motivated examples to demonstrate the efficacy of our software package on diverse representative problems that cover time-to-reach optimal control, and HJI differential games in reachability settings in section 7. Sixth, we conclude the paper in section 8. Additional examples, jupyter notebooks, and representative problems are provided in the online package.

3 DIFFERENTIAL GAMES AND REACHABILITY

In this section, we briefly describe reachability theory within the context of the HJ-Isaacs (HJI) equation. Reachable sets in the context of two person games is introduced. We then establish the "viscuous" P.D.E. to the terminal HJI P.D.E.

For a state $\boldsymbol{x} \in \Omega$, fix: $T > t \geq 0$. Suppose that the controls for opposing players $\boldsymbol{P}$ and $\boldsymbol{E}$ in a two-player differential game are given by measurable functions

$$\boldsymbol{u} : [t, T] \to \mathcal{U}, \qquad \boldsymbol{w} : [t, T] \to \mathcal{W} \tag{4}$$

where $\mathcal{U} \in \mathbb{R}^m$ and $\mathcal{W} \subset \mathbb{R}^p$ are compact sets. Let us consider the differential equation,

$$\dot{\boldsymbol{x}}(\tau) = f(\tau, \boldsymbol{x}(\tau), \boldsymbol{u}(t), \boldsymbol{w}(t)), t \leq \tau \leq T, \quad \boldsymbol{x}(t) = \boldsymbol{x}, \tag{5}$$

where $f(\tau, \cdot, \cdot, \cdot)$ is uniformly continuous and $\boldsymbol{x}(\cdot)$ is the unique solution of system (5). Suppose that we associate with the dynamical system (5) the payoff functional

$$\Phi(t; \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = \int_t^T \boldsymbol{l}(\tau, \boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{w}(\tau)) d\tau + \boldsymbol{g}(\boldsymbol{x}(T)), \tag{6}$$

where $\boldsymbol{g} : \mathbb{R}^n \to \mathbb{R}$ satisfies

$$|\boldsymbol{g}(\boldsymbol{x})| \leq k_1, \qquad |\boldsymbol{g}(\boldsymbol{x}) - \boldsymbol{g}(\hat{\boldsymbol{x}})| \leq k_1 |\boldsymbol{x} - \hat{\boldsymbol{x}}| \tag{7a}$$

and $\boldsymbol{l} : [0, T] \times \mathbb{R}^n \times \mathcal{U} \times \mathcal{W} \to \mathbb{R}$ is the BUC stage cost. We say T is the *terminal time* (it may be infinity!) and the integral, when it does not depend on the control laws, is the *performance index*. The evader is maximizing and pursuer is minimizing the payoff (6). In reachability analysis, the cost-to-go $\boldsymbol{l}(\cdot)$ is taken as zero so that the payoff functional (6) becomes

$$\Phi(t; \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) = \boldsymbol{g}(\boldsymbol{x}(T)). \tag{8}$$

3.1 HJI's Lower Value and its Viscosity Solution

Define

$$\bar{\mathcal{U}} \equiv \{\boldsymbol{u} : [t, T] \to \mathcal{U}\} \text{ and } \bar{\mathcal{W}} \equiv \{\boldsymbol{w} : [t, T] \to \mathcal{W}\} \tag{9}$$

as all control sets for players $\boldsymbol{P}$ and $\boldsymbol{E}$. For controls which agree almost everywhere (a.e.), we are interested in the pursuer's mapping strategy (starting at t) i.e.

$$\beta : \bar{\mathcal{U}}(t) \to \bar{\mathcal{W}}(t) \tag{10}$$

provided for each $t \leq \tau \leq T$ and $\boldsymbol{u}, \hat{\boldsymbol{u}} \in \bar{\mathcal{U}}(t)$; then $\boldsymbol{u}(\bar{t}) = \hat{\boldsymbol{u}}(\bar{t})$ a.e. on $t \leq \bar{t} \leq \tau$ implies $\beta[\boldsymbol{u}](\bar{t}) = \beta[\hat{\boldsymbol{u}}](\bar{t})$ a.e. on $t \leq \bar{t} \leq \tau$. The differential game's *lower value* for a solution $\boldsymbol{x}(t)$ of (5) for a $\boldsymbol{u}(t)$ and $\boldsymbol{w}(t) = \beta[\boldsymbol{u}](\cdot)$ is [12]

$$\begin{aligned} \boldsymbol{v}(\boldsymbol{x}, t) &= \inf_{\beta \in \mathcal{B}(t)} \sup_{\boldsymbol{u} \in \mathcal{U}(t)} \Phi(t, \boldsymbol{x}, \boldsymbol{u}, \beta[\boldsymbol{u}]) \\ &= \inf_{\beta \in \mathcal{B}(t)} \sup_{\boldsymbol{u} \in \mathcal{U}(t)} \boldsymbol{g}\left(\boldsymbol{x}(T)\right). \end{aligned} \tag{11}$$

We say $\boldsymbol{v}(\boldsymbol{x}, t)$ is the lower value function of the differential game. This value function is used in backward reachability analysis, the chief subject of our discourse.

LEMMA 1. *The lower value $\boldsymbol{v}(\boldsymbol{x}, t)$ in* (11) *is the viscosity solution to the lower Hamilton-Jacobi Isaac's equation* (1) *and it has with it the lower Hamiltonian,*

$$\boldsymbol{H}(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}, p) = \max_{\boldsymbol{u} \in \mathcal{U}} \min_{\boldsymbol{w} \in \mathcal{V}} \langle f(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}), p \rangle. \tag{12}$$

where p is the co-state, i.e. the spatial derivative of $\boldsymbol{v}$ w.r.t $\boldsymbol{x}$ and $\langle \cdot, \cdot \rangle$ is the dot product operator between two variables.

PROOF. This proof is given in [12]. □

3.2 Optimal Control, HJI PDE, and Reachability Theory

That which concerns point sets for a dynamical system's behavioral evolution (or trajectory) throughout a phase space[2] and such that the sets satisfy input or state constraints in a least restrictive sense and under a worst-possible disturbance [19] describes reachability. For computational and decidability reasons[3], we restrict our computation scheme over a finite time period. This is a classic reachability theory problem.

Reachability analysis is concerned with finding the set of forward (resp. backward) states that are reachable from a set of initial (resp. target) state sets, *up to a final time* under the worst possible disturbance. This verification problem usually consists in finding a *reachable states set* that lie along the trajectory of the solution to a first order nonlinear P.D.E. Typically, this solution originates from some initial phase $(\boldsymbol{x}_0, t_0)$ up to a state, $\boldsymbol{x}(T)$, at a specified final time T. Once computed, the optimal value function provides a safety certificate and controller defined by the spatial gradients of the value function.

In addition to producing a value function, the optimality principle of dynamic programming on the underlying HJI PDE can be used to generate a minimum time-to-reach (TTR) function. This function maps initial conditions to the minimum time horizon required to reach the target set. This can be computed by "stacking" the zero level sets of the value function as it propagates backwards in time [1, 2, 23].

Backward reachable sets (within a continuous system framework) are those subsets of a phase space whereupon trajectories initialized in some subset of the phase space can reach a so-called target set, $\mathcal{L}(x)$. Mitchell et. al' [24] essential contribution was that viscosity solution of the Cauchy-type HJI PDE (1) or (12) is tantamount to an implicit surface representation of the continuous-time

[2]To avoid the cumbersome phrase "the state $\boldsymbol{x}$ at time t", we will associate the pair $(\boldsymbol{x}, t)$ with the *phase* of the system for a state $\boldsymbol{x}$ at time t. We associate the Cartesian product of Ω and the space $T = \mathbb{R}^1$ of all time values as the *phase space* of $\Omega \times T$.
[3]We say a reachability problem is decidable if we can compute all the states that can be reached from an initial condition in a finite number of steps.

backward reachable set. This formulation is amenable to modern large-scale problems because level set methods can be used in computing the target set as the viscosity solution is numerically continuous everywhere and well-posed. In backward reachability analysis, the ordering of time indices is reversed so that instead of having $t \in [0, T]$, we shall henceforth have $t \in [-T, 0]$.

3.3 The Backward Reachable (Target) Set

Suppose that the goal for an agent moving over a phase space is to reach a region at the end of a time horizon. We could leverage the terminal cost $g(\cdot)$ in (12) and impose constraints

$$|g(0; \boldsymbol{x})| \leq k, \ \ |g(0; \boldsymbol{x}) - g(t; \hat{\boldsymbol{x}}) \ | \leq k|\boldsymbol{x} - \hat{\boldsymbol{x}} \ | \tag{13}$$

for constant k and all $-T \leq t \leq 0$, $\hat{\boldsymbol{x}}, \ \boldsymbol{x} \in \mathbb{R}^n$. The set

$$\mathcal{L}_0 = \{\boldsymbol{x} \in \bar{\Omega} \,|\, g(0; \boldsymbol{x}) \leq 0\}, \tag{14}$$

is called the *target set* (otherwise referred to as the backward reachable set) in the phase space $\Omega \times \mathbb{R}$ (proof in [24]). This target set can represent the failure set (to avoid) or a goal set (to reach) in the state space. Note that the target set, $\mathcal{L}_0$, is a closed subset of $\mathbb{R}^n$ and is in the closure of Ω. Typically $\mathcal{L}_0$ is user-defined, and in level set methods, $g(x)$ is a signed distance function – negative inside the target set and positive elsewhere.

3.4 The Backward Reachable Tube

It is often desirable to consider all conditions under which trajectories of the system may enter a user-defined target set. This could be desirable in goal-regions of the phase space (safe sets) or undesirable configurations (unsafe sets).

For the safety problem setup in (11), we can define the corresponding *robustly controlled backward reachable tube* [23] as the closure of the open set

$$\begin{aligned}\mathcal{L}([\tau, 0], \mathcal{L}_0) = \{\boldsymbol{x} \in \Omega \,|\, \exists\, \beta \in \bar{\mathcal{W}}(t) \,\forall\, \boldsymbol{u} \in \mathcal{U}(t), \exists \\ \tau \in [-T, 0], \boldsymbol{\xi}(\bar{t}) \in \mathcal{L}_0\}.\end{aligned} \tag{15}$$

Read: The set of states from which the strategies of $\boldsymbol{P}$ and for all controls of $\boldsymbol{E}$ imply that we *reach and remain in the target set* in the interval $[-T, 0]$. Following Lemma 2 of [24], the states in the reachable set admit the following properties w.r.t the value function $\boldsymbol{v}$:

$$\boldsymbol{x}(t) \in \mathcal{L}(\cdot) \implies \boldsymbol{v}(\boldsymbol{x}, t) \leq 0, \tag{16a}$$

$$\boldsymbol{v}(\boldsymbol{x}, t) \leq 0 \implies \boldsymbol{x}(t) \in \mathcal{L}(\cdot). \tag{16b}$$

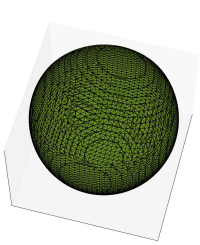
Player $\boldsymbol{P}$ is minimizing (the game's termination time c.f. (14)), seeking to drive system trajectories into the unsafe set; and player $\boldsymbol{E}$ is maximizing (the game's termination time) i.e. is seeking to avoid the unsafe set[4].

4 GEOMETRY OF IMPLICIT SURFACES AND LAYOUTS

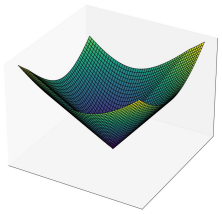
In this section, we discuss how implicit surface functions are constructed, stored on local memory and how they are transferred to GPUs. Throughout, links to `api's`, `routines`, and `subroutines` are highlighted in `blue text` (with a working hyperlink) and we use code snippets in Python to illustrate API calls when it's convenient.

In what follows in this section, at issue are functions $f(x)$ for points x that do not explicitly describe but rather imply the existence of f. This representation is attractive since it requires less

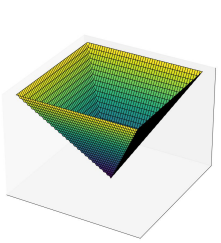
[4]For the goal-satisfaction (or *liveness*) problem setups, the strategies are reversed and the backward reachable tube are the states from which the evader $\boldsymbol{E}$ can successfully reach the target set despite worst-case efforts of the pursuer $\boldsymbol{P}$.



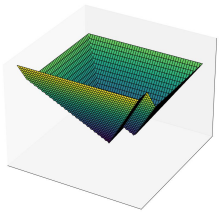
(a) Sphere



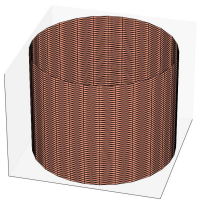
(b) Union of two spheres



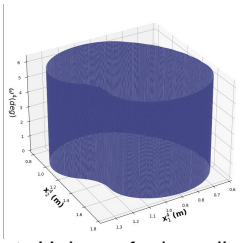
(c) Rectangle



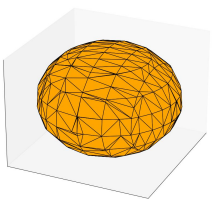
(d) Union of 2 rectangles



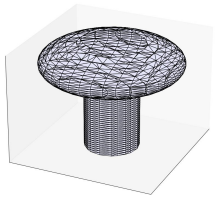
(e) Cylinder



(f) Union of six cylinders



(g) 3D Ellipsoid



(h) Sphere+Cylinder intersect.

Fig. 1. Examples of implicitly constructed zero levelsets of interfaces of geometric primitives along with Boolean operations on 2D and 3D Cartesian grids. Zero level set of **(a)** a sphere on a 3D grid; **(b)** union of two 3D spheres implicitly constructed on a 2D grid; **(c)** a rectangle on a 2D grid; **(d)** the union of rectangles on a 2D grid; **(e)** a cylinder on a 3D grid; **(f)** the union of multiple cylinders on a 3D grid; **(g)** an ellipsoid on a 3D grid; **(h)** Intersection of a sphere and a cylinder on a 3D grid.

number of points to represent a function than explicit forms. Let us describe the representation of data we employ in what follows.

4.1 Grids Layout

Fundamental to implicit surface representations are Cartesian grids in our library. Packages that implement 'grid' data structures are in the folder `Grids`. Grid $\mathbf{g}$ is `created` by specifying minimum, $\mathbf{g}_{min}$, and maximum axes bounds, $\mathbf{g}_{max}$, along every Cartesian coordinate axes n (see lines 3 and 4) of Listing 1; a desired number of discrete points N is passed to the grid data structure – specifying the number of grid nodes and the grid spacing in each dimension as (line 5) listed in Listing 1. On line 5, a grid data structure is constructed and all input parameters to the API are checked for consistency. The contents of the grid data structure are fully described in Definition 1.

```
1 from math import pi
2 import numpy as np
3 gmin = np.array((-5, -5, -pi)) // lower corner
4 gmax = np.array((5, 5, pi)) // upper corner
5 N = 41*ones(3, 1) // number of grid nodes
6 pdim = 3; // periodic boundary condition, dim 3
7 g = createGrid(gmin, gmax, N, pdim)
```

Listing 1. Creating a three-dimensional grid.

DEFINITION 1 (GRID DATA STRUCTURE). *A grid* `data structure`, $\mathbf{g}$, *(implemented in Listing 1) has the following fields: (i) discretized nodes of the state(s) x in* (3)*, denoted as 1-D vectors* $\mathbf{g}$*.vs; (ii) given the 1-D vectors* $\mathbf{g}$*.vs, an n-dimensional array of coordinates over n-dimensional grids*

is computed with matrix-based indexing; this generates a mesh for all state nodal points on the grid $\boldsymbol{g}.xs$ *as a list across all the dimensions of the grid; (iii) grid dimension* $\boldsymbol{g}.dim$*, denoting the number of Cartesian axes needed for representing the state* x[5]*; (iv) boundary conditions of the relavant HJ equation to be solved are grafted in by populating the corresponding grid dimension with ghost cells (to be introduced shortly).*

4.2 Implicit Surface Representations: Levelsets

For an implicit surface representation of a geometric function, we treat the coordinates as functional arguments instead of functional values using a fixed level set of continuous function $\boldsymbol{v} : \mathbb{R}^n \to \mathbb{R}$. We use signed distance functions to represent moving fronts throughout. When the signed distance function is not numerically possible, we describe where the implicit surface representations are smeared out in every routines' `documentation`.

The query points for moving interfaces are grid point sets of the computational domain described by implicit geometric primitives such as spheres, cylinders, ellipsoids and even polyhedrons such as icosahedrons. All of these are contained in the folder `InitialConditions` on our projects page.

Suppose that the zero levelset of an implicit surface $\boldsymbol{v}(x,t)$ is defined as $\Gamma = \{x : \boldsymbol{v}(x) = 0\}$ on a grid $G \in \mathbb{R}^n$, where n denotes the number of dimensions. Our representation of Γ on G generalizes a row-major layout. An example representation of an ellipsoid on a three-dimensional grid is illustrated in Listing 2.

```
1 e = (g.xs[0])**2 // ellipsoid nodal points
2 e += 4.0*(g.xs[1])**2
3 if g.dim==3:
4   data += (9.0*(grid.xs[2])**2)
5 e -= radius // radius=major axis of ellipsoid
```

Listing 2. An `ellipsoid` as a signed distance function.

4.3 Calculus on Implicit Function Representations

Geometrical operations on implicitly defined functions carries through in the package as follows.

Suppose that $\boldsymbol{v}_1(x)$ and $\boldsymbol{v}_2(x)$ are two signed distance representations, then the union of the interior of the two functions is simply $\min(\boldsymbol{v}_1(x), \boldsymbol{v}_2(x))$ (example illustrations in Fig. 1 a, c, d and h). The intersection of the interior of two signed distance functions is generated by $\max(\boldsymbol{v}_1(x), \boldsymbol{v}_2(x))$ (example illustrations in Fig. 1). The complement of a function is found by negating its signed distance function i.e. $-\boldsymbol{v}(x)$. The resultant function as a result of the subtraction of the interior of one signed distance function $\boldsymbol{v}_2$ from the another one, say, $\boldsymbol{v}_1$ is defined $\max(\boldsymbol{v}_1(x), -\boldsymbol{v}_2(x))$. All of these are implemented in the module `shapeOps`.

5 SPATIAL DISCRETIZATION: UPWINDING

Monotone solutions to the levelset equation, as introduced by Crandall and Lions [6] are attractive but they are at most first-order accurate and tend to be dissipative for most practical applications. In this section, we discuss higher-order upwinding schemes that mimic high-order essentially non-socillatory (ENO) schemes for computing the spatial derivatives $\boldsymbol{v}_x$ for the numerical viscosity solutions to levelset PDE's of the *Eulerian form* (introduced in (17)). Codebases for procedures herewith described are in the folder `SpatialDerivatives`.

Using the Eulerian form of the levelset equation,

$$\boldsymbol{v}_t + \boldsymbol{F} \cdot \nabla \boldsymbol{v} = 0 \tag{17}$$

[5]This parameter is useful when computing signed distance to every nodal point on the state space in the implicit representation of $\boldsymbol{v}$

where $\boldsymbol{F}$ is the speed functionI, the implicit function itself (section 4) is used both to denote and to evolve the interface. Suppose that the interface speed $\boldsymbol{F}$ is a three-vector $[f_x, f_y, f_z]$ on a three-dimensional Cartesian grid, expanding (17) the evolution of the implicit function on the zero levelset yields the Eulerian form

$$\boldsymbol{v}_t + f_x \boldsymbol{v}_x + f_y \boldsymbol{v}_y + f_z \boldsymbol{v}_z = 0 \tag{18}$$

of the interface evolution given that the interface encapsulates the implicit representation $\boldsymbol{v}$. In our implementations, we define $\boldsymbol{v}$ throughout the computational domain Ω. However, narrow band methods [30] that only contain the interface can be implemented as well so that memory is saved while the front is being tracked. -

5.1 Spatial Derivatives by Upwinding

Let us first define the following differencing schemes

$$D^- \boldsymbol{v} = \frac{\partial \boldsymbol{v}}{\partial x} \approx \frac{\boldsymbol{v}_{i+1} - \boldsymbol{v}_i}{\Delta x}, D^+ \boldsymbol{v} \approx \frac{\boldsymbol{v}_i - \boldsymbol{v}_{i-1}}{\Delta x}. \tag{19}$$

Suppose that $\boldsymbol{v}$ and its speed $\boldsymbol{F}$ are defined over a domain Ω (this is the Cartesian grid in our representation). Using the forward Euler method, the levelset equation (18) becomes

$$\frac{\boldsymbol{v}^{n+1} - \boldsymbol{v}^n}{\Delta t} + f_x^n \boldsymbol{v}_x^n + f_y^n \boldsymbol{v}_y^n + f_z^n \boldsymbol{v}_z^n = 0. \tag{20}$$

Now, suppose that we are on a one-dimensional surface and around a grid point i, then given that f^n may be spatially varying the equation in the foregoing evaluates to

$$\frac{\boldsymbol{v}_i^{n+1} - \boldsymbol{v}_i^n}{\Delta t} + f_i^n (\boldsymbol{v}_x)_i^n = 0 \tag{21}$$

where $(\boldsymbol{v}_x)_i$ denotes the spatial derivative of $\boldsymbol{v}$ w.r.t x at the point i.

5.2 First-order accurate discretization

If $f_i > 0$, the values of $\boldsymbol{v}$ are traversing from left to right so that in order to update $\boldsymbol{v}$ at the end of the next time step, we must look to the left (going by the method of characteristics [26, §3.1]) and vice versa if $f_i < 0$. We therefore follow the standard upwinding method by using (19): we approximate $\boldsymbol{v}_x$ with $D^- \boldsymbol{v}$ whenever $f_i > 0$ and we approximate $\boldsymbol{v}_x$ with $D^+ \boldsymbol{v}$ whenever $f_i < 0$. No approximation is needed when $f_i = 0$ since $f_i(\boldsymbol{v}_x)_i)$ vanishes. This discretization scheme is accurate within $O(\Delta x)$ given the first order accurate approximations $D^- \boldsymbol{v}$ and $D^+ \boldsymbol{v}$. We have followed the naming convention in [22] and in our `SpatialDerivatives` folder, we name this function `upwindFirstFirst`.

5.3 ENO Polynomial Interpolation of Solutions

Using a divided difference table, essentially nonoscillatory (ENO) polynomial interpolation of the discretization [29] of the levelset equation are known to generate improved numerical approximations to $D^- \boldsymbol{v}$ and $D^+ \boldsymbol{v}$.

Suppose that we choose a uniform mesh discretization Δx. Define the zeroth divided differences of $\boldsymbol{v}$ at the grid nodes i as

$$D_i^0 \boldsymbol{v} = \boldsymbol{v}_i, \tag{22}$$

and the first, second, and third order divided diferences of $\boldsymbol{v}$ as the midway between grid nodes i.e.

$$D^1_{i+1/2}\boldsymbol{v} = \frac{D^0_{i+1}\boldsymbol{v} - D^0_i\boldsymbol{v}}{\Delta x},\ D^2_i\boldsymbol{v} = \frac{D^1_{i+1/2}\boldsymbol{v} - D^1_{i-1/2}\boldsymbol{v}}{2\Delta x},$$
$$D^3_{i+1/2}\boldsymbol{v} = \frac{D^2_{i+1}\boldsymbol{v} - D^2_i\boldsymbol{v}}{3\Delta x}. \tag{23}$$

Then, an essentially non-oscillating polynomial of the form

$$\boldsymbol{v}(x) = Q_0(x) + Q_1(x)!Q_2(x) + Q_3(x) \tag{24}$$

can be constructed. In this light, the backward and forward spatial derivatives of $\boldsymbol{v}$ w.r.t x at grid node i is found in terms of the derivatives of the coefficients $Q_i(x)$ in the foregoing i.e.

$$\boldsymbol{v}_x(x_i) = Q_1'(x_i) + Q_2'(x_i) + Q_3'(x_i). \tag{25}$$

Define $k = i - 1$ and $k = i$ for $\boldsymbol{v}_x^-$ and $\boldsymbol{v}_x^+$ respectively. Then the first order accurate polynomial interpolation is essentially

$$Q_1(x) = (D^1_{k+1/2}\boldsymbol{v})(x - x_i),\ Q_1'(x_i) = D^1_{k+1/2}\boldsymbol{v} \tag{26}$$

i.e. first-order upwinding.

We follow [26]'s recommendation in avoiding interpolating near large oscillations in gradients. Therefore, we choose a constant c such that

$$c = \begin{cases} D^2_k\boldsymbol{v} & \text{if } |D^2_k\boldsymbol{v}| \le |D^2_{k+1}\boldsymbol{v}| \\ D^2_{k+1}\boldsymbol{v} & \text{otherwise} \end{cases} \tag{27}$$

so that

$$Q_2(x) = c(x - x_k)(x - x_{k+1}), \tag{28a}$$
$$Q_2'(x_i) = c(2i - 2k - 1)\Delta x \tag{28b}$$

is the second-order accurate upwinding solution for the polynomial interpolation. This is implemented as `upwindFirstENO2` in the `SpatialDerivatives` folder.

To obtain a third-order accurate solution, we choose $c^\star$ as follows

$$c^\star = \begin{cases} D^3_{k^\star+1/2} & \text{if } |D^3_{k^\star+1/2}\boldsymbol{v}| \le |D^3_{k^\star+3/2}\boldsymbol{v}| \\ D^3_{k^\star+3/2}\boldsymbol{v} & \text{if } |D^3_{k^\star+1/2}\boldsymbol{v}| > |D^3_{k^\star+3/2}\boldsymbol{v}|. \end{cases} \tag{29}$$

Whence, we have

$$Q_3(x) = c^\star(x - x_{k^\star})(x - x_{k^\star} + 1)(x - x_{k^\star} + 2) \tag{30a}$$
$$Q_3'(x_i) = c^\star(3(i - k^\star)^2 - 6(i - k^\star) + 2)(\Delta x)^2 \tag{30b}$$

for the third-order accurate correction to the approximated upwinding scheme (24). This is implemented as a routine in `upwindFirstENO3aHelper` and called as `upwindFirstENO3` in the `SpatialDerivatives` folder.

5.4 HJ Weighted Essentially Nonoscillatory Solutions

Here, we focus on weighted ENO (WENO) schemes with the same stencil as the third-order ENO scheme but with accuracy reaching as high as fifth-order in the smooth parts of the solution. Results here presented closely follow the presentation of Jiang and Peng in [16]. These WENO schemes approximate spatial derivatives at integer grid points as opposed to at half-integer grid values as we did in the ENO schemes in the previous section.

The third-order accurate ENO scheme essentially employs one of three substencils on a grid, namely $\{i-3, i-2, \cdots, i\}$, $\{i-2, i-1, \cdots, i+1\}$, and $\{i, \cdots, i+3\}$ on the stencils range $\{i-3, i-2, \cdots, i+3\}$ in calculating spatial derivatives for $\boldsymbol{v}$.

Suppose that the spatial derivative $\boldsymbol{v}_x$ is to be found using the left-leaning substencil: $\{i-3, i-2, \cdots, i\}$, then the third-order ENO scheme chooses one from

$$\boldsymbol{v}_{x,i}^{-,0} = \frac{1}{3}D^+\boldsymbol{v}_{i-3} - \frac{7}{6}D^+\boldsymbol{v}_{i-2} + \frac{11}{6}D^+\boldsymbol{v}_{i-1} \tag{31a}$$

$$\boldsymbol{v}_{x,i}^{-,1} = -\frac{1}{6}D^+\boldsymbol{v}_{i-2} + \frac{5}{6}D^+\boldsymbol{v}_{i-1} + \frac{1}{3}D^+\boldsymbol{v}_i \tag{31b}$$

$$\boldsymbol{v}_{x,i}^{-,2} = -\frac{1}{3}D^+\boldsymbol{v}_{i-1} + \frac{5}{6}D^+\boldsymbol{v}_i - \frac{1}{6}D^+\boldsymbol{v}_{i+1} \tag{31c}$$

where $\boldsymbol{v}_{x,i}^{-,p}$ denotes the third-order p'th substencil to $\boldsymbol{v}_x(x_i)$ for $p = 0, 1, 2$. The WENO approximation to $\boldsymbol{v}_x(x_i)$ leverages a convex weighted average of the three substencils so that

$$\boldsymbol{v}_{x,i}^{-} = w_0\boldsymbol{v}_{x,i}^{-,0} + w_1\boldsymbol{v}_{x,i}^{-,1} + w_2\boldsymbol{v}_{x,i}^{-,2}. \tag{32}$$

In smooth regions of the phase space, $w_0 = 0.1$, $w_1 = 0.6$ and $w_2 = 0.3$ yield the optimally accurate fifth order WENO approximation, we have for $\boldsymbol{v}_{x,i}^{-}$

$$\begin{aligned}\frac{1}{30}D^+\boldsymbol{v}_{i-3} - \frac{13}{60}D^+\boldsymbol{v}_{i-2} + \frac{47}{60}D^+\boldsymbol{v}_{i-1} \\ + \frac{9}{20}D^+\boldsymbol{v}_i - \frac{1}{20}D^+\boldsymbol{v}_{i+1}\end{aligned} \tag{33}$$

the fifth-order approximation $\boldsymbol{v}_x(x_i)$ and provides the smallest truncation error on a six-point stencil.

To account for weights in non-smooth regions, however, the smoothness of the stencils (31) can be estimated as recommended in [26, §3.4] so that if

$$\alpha_1 = 0.1/(\sigma_1 + \epsilon)^2,\ \alpha_2 = 0.6/(\sigma_2 + \epsilon)^2,\ \alpha_3 = 0.1/(\sigma_3 + \epsilon)^2 \tag{34}$$

for

$$\begin{aligned}\sigma_1 = \frac{13}{12}(D^+\boldsymbol{v}_{i-3} - 2D^+\boldsymbol{v}_{i-2} + D^+\boldsymbol{v}_{i-1})^2 \\ + \frac{1}{4}(D^+\boldsymbol{v}_{i-3} - 4D^+\boldsymbol{v}_{i-2} + 3D^+\boldsymbol{v}_{i-1})^2,\end{aligned} \tag{35a}$$

$$\begin{aligned}\sigma_2 = \frac{13}{12}(D^+\boldsymbol{v}_{i-2} - 2D^+\boldsymbol{v}_{i-3} + D^+\boldsymbol{v}_i)^2 \\ + \frac{1}{4}(D^+\boldsymbol{v}_{i-2} - 2D^+\boldsymbol{v}_i)^2,\end{aligned} \tag{35b}$$

$$\begin{aligned}\sigma_3 = \frac{13}{12}(D^+\boldsymbol{v}_{i-1} - 2D^+\boldsymbol{v}_i + D^+\boldsymbol{v}_{i+1})^2 \\ + \frac{1}{4}(3D^+\boldsymbol{v}_{i-1} - 4D^+\boldsymbol{v}_i + D^+\boldsymbol{v}_{i+1})^2,\end{aligned} \tag{35c}$$

and

$$\begin{aligned}\epsilon = 10^{-6}\max\{D^+\boldsymbol{v}_{i-3}, D^+\boldsymbol{v}_{i-2}, D^+\boldsymbol{v}_{i-1}, D^+\boldsymbol{v}_i, \\ D^+\boldsymbol{v}_{i+1}\} + 10^{-99}\end{aligned} \tag{36}$$

then, we may define the weights for the WENO scheme as

$$w_1 = \alpha_1/\sum_{i=1}^{3}\alpha_i,\ \ w_2 = \alpha_2/\sum_{i=1}^{3}\alpha_i,\ \ w_3 = \alpha_3/\sum_{i=1}^{3}\alpha_i. \tag{37}$$

which well approximates the optimal weights $w_0 = 0.1$, $w_1 = 0.6$ and $w_2 = 0.3$ for decently smooth σ_k that can be dominated ny ϵ. This is implemented as a routine in `upwindFirstWENO5a` and called as `upwindFirstWENO5`.

5.5 Lax-Friedrichs Monotone Difference Schemes

We now describe a convergent monotone difference spatial approximation scheme for scalar conservation laws of the form

$$\begin{aligned} \boldsymbol{v}_t + \sum_{i=1}^{N} f_i(\boldsymbol{v})_{x_i} = 0 \text{ for } t > 0, \boldsymbol{x} = (x_1, \cdots, x_N) \in \mathbb{R}^n \\ \boldsymbol{v}(\boldsymbol{x}, 0) = \boldsymbol{v}_0(\boldsymbol{x}), \text{ for } \boldsymbol{x} \in \mathbb{R}^n \end{aligned} \tag{38}$$

Suppose that $N = 1$, let us define $\lambda_x = \Delta t/\Delta x$, $\Delta_x^+ = \boldsymbol{v}_{j+1} - v_j$, and $\Delta_x^- = \boldsymbol{v}_j - \boldsymbol{v}_{j-1}$. Then at the nth time step, the Lax-Friedrichs scheme is [7]

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \frac{\lambda_x}{2}\Delta_x^0 f(\boldsymbol{v}_j^n) + \frac{1}{2}\Delta_x^+\Delta_x^- \boldsymbol{v}_j^n. \tag{39}$$

Furthermore, if we define the flux on the state space as

$$g(\boldsymbol{v}_j, \boldsymbol{v}_{j-1}) = \frac{f(\boldsymbol{v}_j) + f(\boldsymbol{v}_{j-1})}{2} - \frac{1}{2}\lambda_x(\boldsymbol{v}_j - \boldsymbol{v}_{j-1}), \tag{40}$$

we may write

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \lambda_x^+(\boldsymbol{v}_j, \boldsymbol{v}_{j-1}). \tag{41}$$

The Lax-Friedrichs scheme is monotone on the interval $[a, b]$ if the CFL condition

$$\lambda_x \max_{a \le \boldsymbol{v} \le b} |f'(\boldsymbol{v})| \le 1 \tag{42}$$

for $(a, b) > 0$ and the upwind differencing scheme for a nondecreasing f is

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \lambda_x\Delta_x^+ f(\boldsymbol{v}_{j-1}^n). \tag{43}$$

For a nonincreasing f, we have

$$\boldsymbol{v}_j^{n+1} = \boldsymbol{v}_j^n - \lambda_x\Delta_x^+ f(\boldsymbol{v}_j^n). \tag{44}$$

The rest of the implementation we provide closely follows that of [22], implemented as `termLaxFriedrichs`.

6 TEMPORAL DISCRETIZATION: METHOD OF LINES

Here, we describe further improvements on the numerical derivatives of HJ equations by further improving the fifth order accurate HJ WENO schemes presented in section 5. We adopt the *method of lines* (MOL) used in converting the time-dependent PDE's to ODEs.

In the MOL, the spatial derivatives in the PDE are replaced with algebraic approximations (in our case, one of the ENO schemes earlier presented) so that spatial derivatives no longer explicitly depend on spatial independent variables. Whence, only time, the initial value variable, is left so that we end up with a system of ordinary differential equations (ODEs) that closely approximate the original PDE.

Our presentation here follows the total variation diminishing (TVD) Runge Kutta (RK) schemes with Courant-Friedrichs-Lewy (CFL) conditioning imposed for stability as presented in [28] and implemented in `MATLAB®` in [22].

6.1 Higher-Order TVD-RK Time Discretizations

To adopt the method of lines, the N-dimensional levelset representation of v is first rolled into a 1-D vector and an adaptive integration step size, Δt, is chosen to guarantee stability following the recommendation in [32]. The forward Euler algorithm thus becomes

$$v(x, t + \Delta t) = v(x, t) + \Delta t \Upsilon(x, v(x, t)) \tag{45}$$

where Υ is now the function to be integrated.

A standard MOL can then be applied for the integration similar to ODEs (we have followed [22]'s code layout to provide consistency for MATLAB users). Since the details of the implementation are described in [22], we here describe the function arguments and describe call signatures.

We implement TVD-RK MOL schemes up to third-order accurate forward Euler integration schemes and the calling signature is as described in Listing 3.

```
odeCFLx(schemeFunc, tspan, y0, options, schemeData)
```

Listing 3. CFL-constrained method of lines routines.

where x could be one of 1, 2, or 3 to indicate first-order accurate, second-order accurate, or third-order accurate TVD-RK scheme. `schemeFunc` is typically a one of the Lax-Friedrichs approximation routines (implemented as `termLaxFriedrichs`) in the folder `ExplicitIntegration/Term` that approximates the HJ equation based on dissipation functions (to be shortly introduced).

The first-order accurate TVD (it is total variation bounded [TVB] actually) together with the spatial discretization used for the PDE is equivalent to the forward Euler method. We implement this as `odeCFL1`.

The second-order accurate TVD-RK scheme follows the RK scheme by evolving the Euler step to $t^n + \Delta t$,

$$\frac{v^{n+1} - v^n}{\Delta t} + F^n \cdot \nabla v^n = 0. \tag{46}$$

A following Euler step to $t^n + 2\Delta t$ follows such that

$$\frac{v^{n+2} - v^{n+1}}{\Delta t} + F^{n+1} \cdot \nabla v^{n+1} = 0 \tag{47}$$

before a convex combination of the initial value function and the result of the preceding Euler steps is taken in the following averaging step

$$v^{n+1} = \frac{1}{2}\{v^n + v^{n+2}\}. \tag{48}$$

The equation in the foregoing produces the second-order accurate TVD approximation to v at $t^n + \Delta t$, implemented as `odeCFL2`.

With the third-order accurate TVD-RK scheme, the first two advancements in forward Euler schemes are computed but with a different averaging scheme

$$v^{n+1/2} = \frac{1}{4}\{3v^n + v^{n+2}\} \tag{49}$$

which averages the previous two solutions at $t^n + \frac{1}{2}\Delta t$. The third Euler advancement step to $t^n + \frac{3}{2}\Delta t$ is

$$\frac{v^{n+\frac{3}{2}} - v^{n+\frac{1}{2}}}{\Delta t} + F^{n+\frac{1}{2}} \cdot \nabla v^{n+\frac{1}{2}} = 0 \tag{50}$$

together with the avergaing scheme

$$\boldsymbol{v}^{n+1} = \frac{1}{3}\{\boldsymbol{v}^{n} + 2\boldsymbol{v}^{n+\frac{3}{2}}\} \tag{51}$$

to produce a third-order accurate approximation to $\boldsymbol{v}$ at time $t^n + \Delta t$, implemented as `odeCFL3`.

7 EXAMPLES AND NUMERICAL EXPERIMENTS

In this section, we will present problems motivated by real-world scenarios and amend them to a form where their solutions can be computed with our `LevelSetPy` toolbox. The problems that we consider belong in `transport`, `differential games`, and `time-to-reach` problem classes.

For the game theory scenarios that we present, each trajectory optimization episode within a game may be characterized as a pursuit *game*, Γ. And by a game, we do not necessarily refer to a single game, but rather a *collection of games*, $\Upsilon = \{\Gamma_1, \cdots, \Gamma_g\}$. Such a game terminates when *capture* occurs, that is the distance between players falls below a predetermined threshold. Each player in a game shall constitute either a pursuer ($\boldsymbol{P}$) or an evader ($\boldsymbol{E}$). Let the cursory reader not interpret $\boldsymbol{P}$ or $\boldsymbol{E}$ as controlling a single agent. In our various numerical experiments, we are poised with one or several pursuers (enemies) or evaders (peaceful citizens). However, when $\boldsymbol{P}$ or $\boldsymbol{E}$ governs the behavior of but one agent, these symbols will denote the agent itself.

We must settle upon how best should $\boldsymbol{P}$ pursue $\boldsymbol{E}$. Here, at every time instant, $\boldsymbol{P}$ possesses knowledge of his own and that of $\boldsymbol{E}$'s position so that $\boldsymbol{P}$ knows how to regulate its various controlling variables with respect to $\boldsymbol{E}$'s motion in an optimal fashion. This follows the mathematical layout in section 3.

The rest of this section introduces three different representative examples where real-world problems are adopted and amended to the HJ PDE form and whose solution we seek to recover with the library presented. First, we present a rocket pursuit-evasion game where the goal is for the evader to guard a territory and the pursuer's goal is to penetrate the boundary of the territory to be guarded. Second, we describe a double integral dynamical system[6] and provide the numerical enumeration of the solution to the analytical time to reach problem. Third, we describe a collective behavior system and we provide a mathematical abstraction that allows the computation of the collision-avoidance system that may then be used in a runtime assurance (RTA) safety-critical controller[7].

7.1 The Rocket Pursuit-Evasion Terminal Value Differential Game

We adopt the rocket problem of Dreyfus [10]: the goal is to launch a rocket in fixed time to a desired altitude, given a final vertical velocity component and a maximum final horizontal component as constraints. We amend the original problem to computing the barrier surface of the *usable part* [15] of the terminal surface using our `LevelSetPy` toolbox.

Therefore, we discard the constraints and turn the problem to a planar differential game between the two rockets. The rockets share similar dynamics in a general sense with one serving as a pursuer $\boldsymbol{P}$ and the other as an evader $\boldsymbol{E}$. We place the evader at the origin in our formulation.

[6]The double integral plant is a simplified abstraction of many real-world force-control system e.g. that obey Newton's second law of motion or the torque-inertia dynamics of a body with rotary dynamics

[7]RTA controllers act intelligently as a safety system between a real-world controller and the system to be controlled by providing a state monitoring system that may be used in intervening on a real-world dynamical system when vulnerabilities to danger become critical.

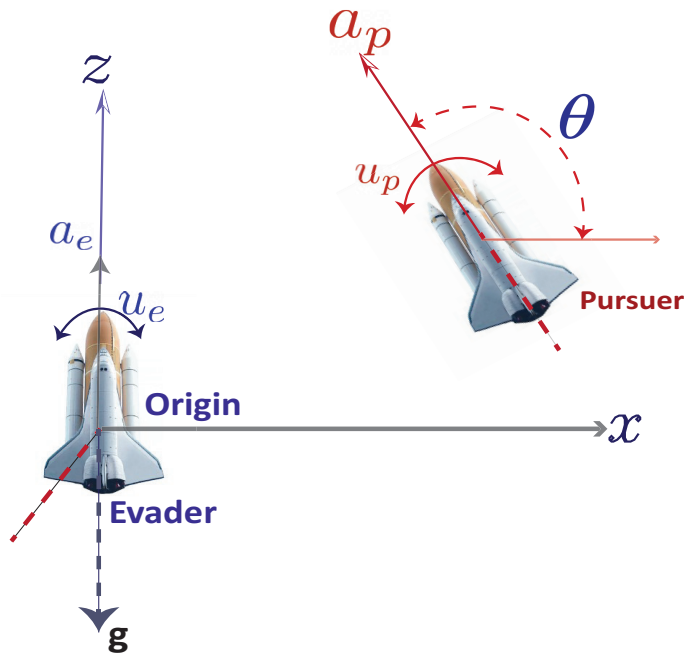


Fig. 2. Motion of two rockets on a Cartesian xz-plane with a thrust inclination in relative coordinates given by $\theta := u_p - u_e$.

Ours is an instance of a terminal differential game. A single rocket's motion is dictated by the following differential equations (under Dreyfus' assumptions):

$$\dot{x}_1 = x_3, \qquad x_1(t_0) = 0; \tag{52a}$$
$$\dot{x}_2 = x_4, \qquad x_2(t_0) = 0; \tag{52b}$$
$$\dot{x}_3 = a\cos u, \qquad x_3(t_0) = 0; \tag{52c}$$
$$\dot{x}_4 = a\sin u - g, \qquad x_4(t_0) = 0; \tag{52d}$$

where, (x_1, x_2) are respectively the horizontal and vertical range of a rocket (in feet), (x_3, x_4) are respectively the horizontal and vertical velocities of a rocket (in feet per second), while a and g are respectively the acceleration and gravitational accelerations (in feet per square second).

This is essentially a pursuit game with the game terminating when *capture* occurs. That is, the distance $\boldsymbol{PE}$ becomes less than a certain pre-specified (scalar) quantity. Being a free endpoint problem, we transform it into a game between two players without the terminal time constraints.

Let the states of $\boldsymbol{P}$ and $\boldsymbol{E}$ be now respectively be denoted by (x_p, x_e). Furthermore, let the rockets be driven by their thrusts (u_p, u_e) respectively in the (x, z)-plane (see Figure 2). We have fixed the range of the rockets in our problem setup so that what is left of the motion of either $\boldsymbol{P}$ or $\boldsymbol{E}$'s is restricted orientation on the (x, z) plane as illustrated in Fig. 2.

Whence, the relevant kinematic equations are (52b) and (52d). The question of what is the "best" outcome must be decided. For the problem at hand, *the payoff*, $\boldsymbol{\phi}$, which is the distance of $\boldsymbol{P}$ from $\boldsymbol{E}$ when capture occurs (denoted as $\|\boldsymbol{PE}\|_2$) shall be chosen as our performance criterion. We say capture occurs when $\|\boldsymbol{PE}\|_2 \leq \boldsymbol{\phi}$ for a pre-specified capture radius, $\boldsymbol{\phi} > 0$. We say $\boldsymbol{P}$ controls u_p and is minimizing $\boldsymbol{\phi}$, and $\boldsymbol{E}$ controls u_e and is maximizing $\boldsymbol{\phi}$.

Our current desideratum is determining if capture can be achieved at all in a “yes-or-no" fashion. That is, we have a finite range over outcomes so that the game at hand assumes Isaac's [15] description of *a game of kind*. We set up the game so that $\boldsymbol{P}$ can achieve as much proximity to the *target set* as much as possible while $\boldsymbol{E}$ is set up to protect the *target set* (see a depiction of the system in Fig. 2). Setting up $\boldsymbol{E}$ to maximize the payoff quantity (54) with the largest possible margin or at least frustrate the efforts of $\boldsymbol{P}$ with minimal collateral damage while the pursuer minimizes the payoff quantity constitutes a terminal value *optimal* differential game: there is no optimal pursuit without an optimal evasion since $\boldsymbol{P}$ and $\boldsymbol{E}$ are both executing motions as they see fit within the parameters of the problem.

We now make the problem amenable to a two-player differential game analysis so that every max and min operations are in the interior of the plane and no sudden changes from extremes are too aggravating in cost (the payoff introduced earlier).

The coordinates of $\boldsymbol{P}$ are freely chosen; however, the coordinates of $\boldsymbol{E}$ are chosen a distance $\boldsymbol{\phi}$ away from (x, z) so that the $\boldsymbol{EP}$ vector's inclination measured counterclockwise from the x axis is θ (this shall serve as the control input). Therefore, we rewrite (52) with $\boldsymbol{P}$'s motion relative to $\boldsymbol{E}$'s along the (x, z) plane so that the relative orientation as shown in Fig. 2 is $\theta = u_p - u_e$. Following the conventions in Fig. 2, the game's relative equations of motion in *reduced space* [15, §2.2] i.e. is $\boldsymbol{x} = (x, z, \theta)$ where $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)$ and $(x, z) \in \mathbb{R}^2$ are

$$\dot{\boldsymbol{x}} = \begin{cases} \dot{x} &= a_p \cos\theta + u_e x, \\ \dot{z} &= a_p \sin\theta + a_e + u_e x - g, \\ \dot{\theta} &= u_p - u_e. \end{cases} \tag{53}$$

The boundary of the *usable part* [15] of the origin-centered circle of radius $\boldsymbol{\phi}$ (we set $\boldsymbol{\phi} = 1.5$ feet in our evaluations) is $\|\boldsymbol{PE}\|_2$. In this sentiment, we find that

$$\boldsymbol{\phi}^2 = x^2 + z^2, \tag{54}$$

and all capture points are specified by

$$\dot{\boldsymbol{\phi}}(\boldsymbol{x}, t) + \min\left[0, \boldsymbol{H}(\boldsymbol{x}, \frac{\partial \boldsymbol{\phi}(\boldsymbol{x}, t)}{\partial \boldsymbol{x}})\right] \leq 0, \tag{55}$$

with the corresponding Hamiltonian

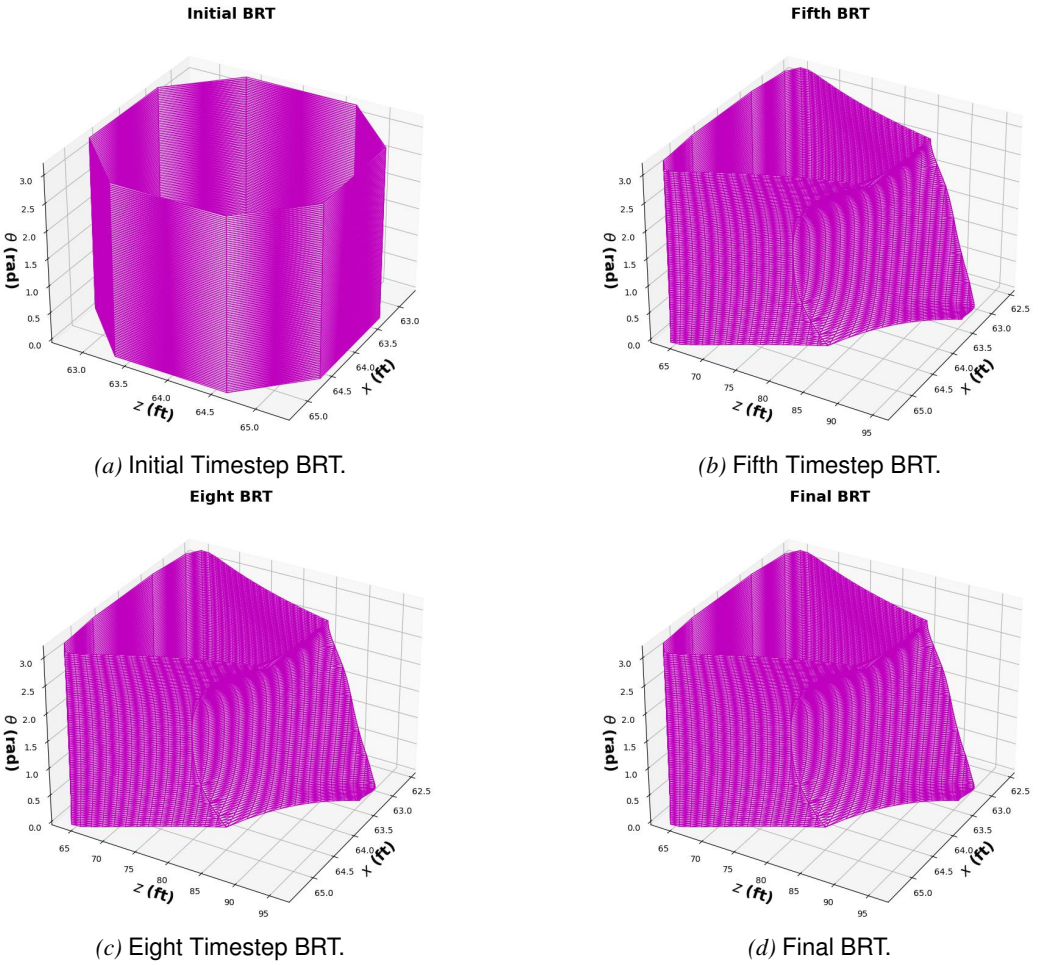
$$\boldsymbol{H}(\boldsymbol{x}, p) = -\max_{u_e \in \mathcal{U}_e} \min_{u_p \in \mathcal{U}_p} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}^T \begin{bmatrix} a_p \cos\theta + u_e x \\ a_p \sin\theta + a_e + u_p x - g \\ u_p - u_e \end{bmatrix}. \tag{56}$$

We must consider the possibilities of behavior by either agent in an all-encompassing fashion in order to know what an outcome may be in the future should either agent execute different controls. Rather than resort to analytical *geometric reasoning*, we may analyze this *game* via a principled numerical simulation. This is what we present next.

We set the linear velocities and accelerations equal to one another i.e. $u_e = u_p$ and $a_e = a_p$ so that the Hamiltonian takes the form

$$\begin{aligned} \boldsymbol{H}(\boldsymbol{x}, p) = &-\cos(u)|ap_1| + \cos(u)|ap_1| - \sin(u)|ap_2| - \\ &\sin(u)|ap_2| + u|p_3| - u|p_3|. \end{aligned} \tag{57}$$

A three-dimensional grid with uniformly spaced grid nodes over an interval $[-64, 64]$ and at a resolution of 100 points per dimension was used in implicitly constructing the *terminal surface* for the values (54).



(a) Initial Timestep BRT.

(b) Fifth Timestep BRT.

(c) Eight Timestep BRT.

(d) Final BRT.

Fig. 3. *(Left to Right)*: Backward reachable tubes (capture surfaces) for the rocket system (cf. Fig. 2) optimized for the paths of slowest-quickest descent in equation (56) at the *(a)* first time step implicitly constructed using the interface method outlined in §4; *(b)* fifth global timestep; *(c)* eight global time step (near convergence to the optimal capture surface) *(d)* final time step. In all, the BRTs were computed using the method outlined in [4, 23, 26]. We set $a_e = a_p = 64ft/sec^2$ and $g = 32ft/sec^2$ as in Dreyfus' original example.

We are interested in the *terminal surface* on the boundary of the open set Ω introduced in (1) and (3). This terminal surface describes regions of the phase space where capture is avoided in our probem definition, and as long as $\boldsymbol{E}$ remains within this *backward reachable tube* (or BRT), $\boldsymbol{P}$ cannot cause damage or exercise an action of deleterious consequence on, say, the territory being guarded by $\boldsymbol{E}$. This eventual terminal represents the final overapproximated (non)-capture surface for the evader.

```
finite_diff_data = {"innerFunc": termLaxFriedrichs,
  "innerData": {"grid": g, "hamFunc": rocket_rel.ham,
  "partialFunc": rocket_rel.dissipation,
  "dissFunc": artificialDissipationGLF,
  "CoStateCalc": upwindFirstENO2},
  "positive": True} // direction of approx. growth
```

Listing 4. HJ ENO2 computational scheme for the rockets.

The Hamiltonian, upwinding scheme, flux dissipation method, and the overapproximation parameter for the essentially nonoscillatory polynomial interpolatory data used in geometrically reasoning about the *target set* is set up as seen in Listing 4. The data structure `finite_diff_data` contains all the routines needed for adding dynamics to the original implicit surface representation of $v(x,t)$. The monotone spatial upwinding scheme used (here `termLaxFriedrichs` described in §5.5) is passed into the `innerFunc` query field. The explicit form of the Hamiltonian (see (58)) is passed to the `hamFunc` query field and the grid described in the foregoing is passed to the `grid` field. We adopted a second-order accurate upwinding scheme together with the `Lax-Friedrichs` approximator. To indicate to the numerical algorithm that we intend to overapproximate the value function as time marches forward, a specify a `True` parameter for the `positive` query field.

Safety is engendered by having the evader respond optimally to the pursuer at various specific time steps t_i. However, we are interested in the entire set, over time, that constitute the safe set. Therefore, we consider computing the safety tube. And because we are computing the tube backward in time (this is done by multiplying the system's Hamiltonian by -1), we compute the backward reachable tube (BRT) [24], under the control strategies of $\boldsymbol{P}$ or $\boldsymbol{E}$.

The BRT is a part of the phase space that constitutes $\Omega \times T$. We would like the backward reachable tube to cover as much of the entire phase space as possible. Thus, we *overapproximate* it. Using our GPU-accelerated levelset toolbox, we compute the *overapproximated* BRT of the game over a time span of $[-2.5, 0]$ seconds over 11 global optimization time steps. The BRTs at representative time steps in the optimization procedure is depicted in Fig. 3.

The initial value function (leftmost inset of Fig. 3) is represented as a (closed) dynamic implicit surface over all point sets in the state space (using a signed distance function) for a coordinate-aligned cylinder whose vertical axes runs parallel to the orientation of the rockets depicted in Fig. 2. This closed and bounded assumption of the target set is a prerequisite of the backward reachable analysis (see [24]). It allows us to include all limiting velocities The two middle capture surfaces indicate the evolution of the capture surface (here the zero levelset) of the target set upon the optimal response of the evader to the pursuer. We reach convergence at the eleventh global optimization timestep (rightmost inset of Fig. 3).

Reachability theory thus affords us an ability to numerically reason about the behavior of these two rockets aforetime in a principled manner. To do this, we have passed relevant parameters to the package as shown in Listing 4 and run a CFL constrained optimization scheme (as in Listing 3) for a finite number of global optimization timesteps. It is global because internally, there are other local spatial and temporal finite differencing scheme that occurs "under the hood" (see 5 and the corresponding codes described).

7.2 Time Optimal Control: The Double Integral Plant

Here, we analyze a time-optimal control problem to determine what admissible control[8] can "transport" the system under consideration to a desired "origin" in the shortest possible time. We consider the double integral plant as an illustrative example of our objective, which is to compute the points in the state space that can reach the origin in finite-time under the influence of a time-optimal controller.

[8]A control law is admissible when its range belongs in the admissible input set where it is bounded.

We shall leverage standard necessary conditions from the principle of optimality [3] to obtain a time-optimal feedback control design; introduce the notion of isochrones and switching surfaces; and discuss the analytic and approximate solutions (with our library) to the time-optimal control problem for a double integrator. We shall conclude the section by comparing the analytic and the overapproximated numerical solution (using the LevelsetPy toolbox) to the time to reach the origin problem.

7.2.1 Dynamics and Problem Setup. The double integrator is controllable, so that open-loop strategies may be employed in driving specific states to the origin in finite time [33]. The plant has the following second-order dynamics

$$\ddot{\boldsymbol{x}}(t) = \boldsymbol{u}(t) \tag{58}$$

and admits bounded control signals $\mid \boldsymbol{u}(t) \mid \leq 1$ for all time t. After a change of variables,we have the following system of first-order differential equations

$$\begin{aligned} \dot{\boldsymbol{x}}_1(t) &= \boldsymbol{x}_2(t), \\ \dot{\boldsymbol{x}}_2(t) &= \boldsymbol{u}(t), \quad \mid \boldsymbol{u}(t) \mid \leq 1. \end{aligned} \tag{59}$$

The *reachability problem we consider is to address the states that can reach a certain point (here, the origin) in the state space in a transient manner*. That is, we would like to find point sets on the state space, at a particular time step, such that we can bring the system to an equilibrium, say, $(0, 0)$.

7.2.2 Time-optimal control scheme. This is an $\boldsymbol{H}$-minimal control problem whereupon we must find the control law that minimizes the Hamiltonian

$$\boldsymbol{H}(\boldsymbol{x}, p) = p_1 \dot{\boldsymbol{x}}_1 + p_2 \dot{\boldsymbol{x}}_2. \tag{60}$$

The necessary optimality condition stipulates that the minimizing control law be

$$\boldsymbol{u}(t) = -\operatorname{sign}\left(p_2(t)\right) \triangleq \pm 1. \tag{61}$$

For the co-states in question, suppose that their initial values (for constants k_1 and k_2) are $p_1(t_0) = k_1$ and $p_2(t_0) = k_2$, only four candidates can serve as time-optimal control sequences i.e. $\{[+1], [-1], [+1, -1], [-1, +1]\}$. On a finite time interval, $t \in [t_0, t_f]$, the time-optimal $\boldsymbol{u}(t)$ is a constant $k \equiv \pm 1$ so that for initial conditions $\boldsymbol{x}_1(t_0) = \xi_1$ and $\boldsymbol{x}_2(t_0) = \xi_2$, it can be verified that state trajectories obey the relations

$$\boldsymbol{x}_1(t) = \xi_1 + \frac{1}{2} k \left(\boldsymbol{x}_2^2 - \xi_2^2\right), \text{ for } t = k\left(\boldsymbol{x}_2(t) - \xi_2\right). \tag{62}$$

The trajectories of (63) traced out over a finite time horizon $t = [-1, 1]$ with *piecewise constant control laws*, $u = \pm 1$ on a state space and under the control laws $\boldsymbol{u}(t) = \pm 1$ is depicted in Fig. 4. Curves with arrows that point upwards denote trajectories under the control law $\boldsymbol{u} = +1$; call these trajectories $\boldsymbol{\gamma}_+$; while the trajectories marked by dashed arrows pointing downward on the curves were executed under $\boldsymbol{u} = -1$; call these trajectories $\boldsymbol{\gamma}_-$.

7.2.3 Analytic Time to Reach the Origin. The *time to reach the origin* $(0, 0)$ from any other pair (x_1, x_2) on the state plane of Fig. 4 *in the shortest possible time* is our approximation problem. This minimum time admits an analytical solution given by [1]

$$t^\star(\boldsymbol{x}_1, \boldsymbol{x}_2) = \begin{cases} \boldsymbol{x}_2 + \sqrt{4\boldsymbol{x}_1 + 2\boldsymbol{x}_2^2} & \text{if } \boldsymbol{x}_1 > \frac{1}{2}\boldsymbol{x}_2|\boldsymbol{x}_2| \\ -\boldsymbol{x}_2 + \sqrt{-4\boldsymbol{x}_1 + 2\boldsymbol{x}_2^2} & \text{if } \boldsymbol{x}_1 < -\frac{1}{2}\boldsymbol{x}_2|\boldsymbol{x}_2| \\ |\boldsymbol{x}_2| & \text{if } \boldsymbol{x}_1 = \frac{1}{2}\boldsymbol{x}_2|\boldsymbol{x}_2|. \end{cases} \tag{63}$$

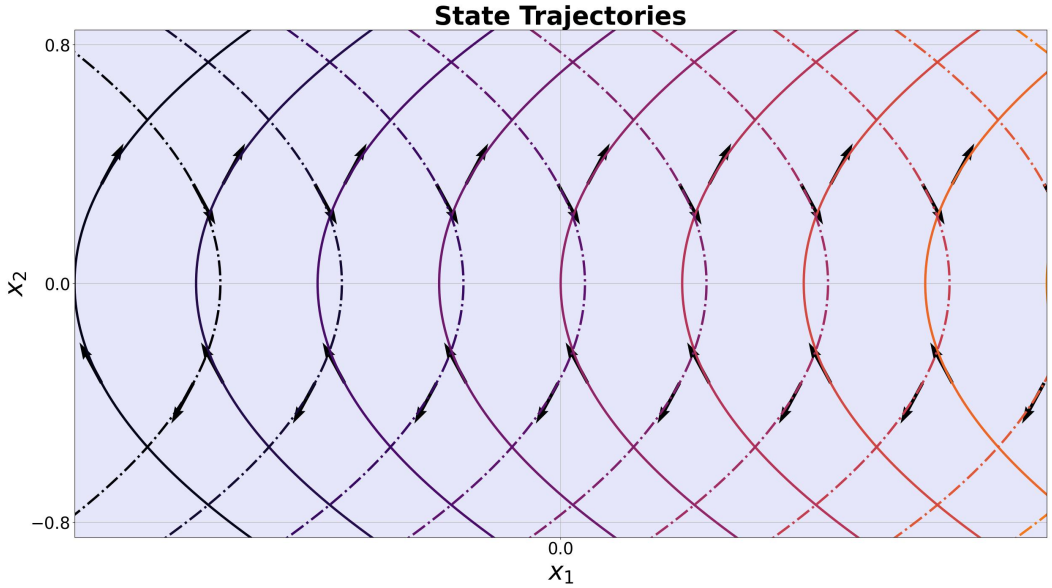


Fig. 4. State trajectories of the double integral plant. The solid curves are trajectories generated for $\boldsymbol{u} = +1$ while the dashed curves are trajectories for $\boldsymbol{u} = -1$.

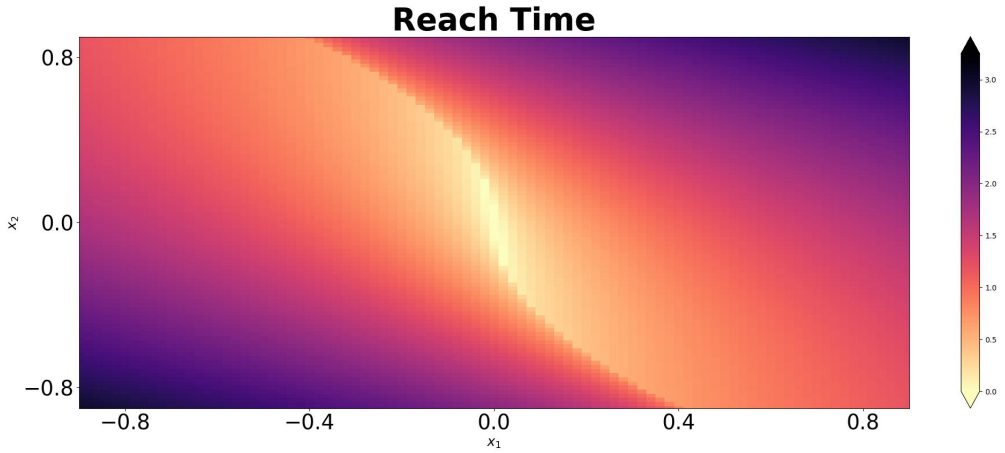


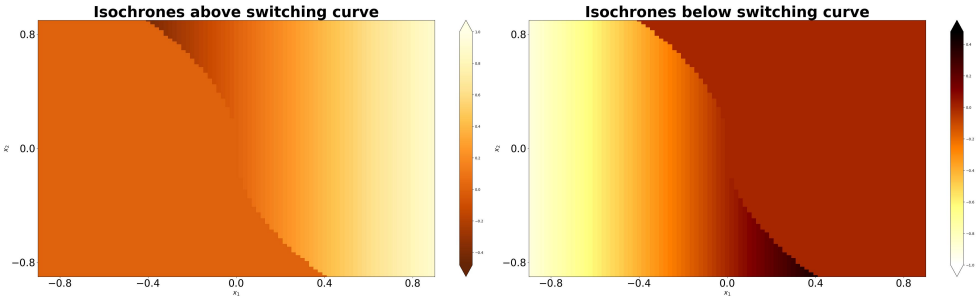
Fig. 5. Analytical time to reach the origin on the state grid, $(\mathbb{R} \times \mathbb{R})$; the switching curve, $\gamma = \gamma_- \cup \gamma_+$, passes through states on $(0, 0)$.

The phase portrait of (64) is shown in Fig. 5. Let us define γ_+ as the locus of all points (x_1, x_2) which can be forced to the origin by $u = +1$ i.e.

$$\gamma_+ = \{(x_1, x_2) : x_1 = \frac{1}{2}x_2^2;\ x_2 \leq 0\} \tag{64}$$

and let γ_- be the locus of all points (x_1, x_2) which can be forced to the origin by $u = -1$ i.e.

$$\gamma_- = \{(x_1, x_2) : x_1 = \frac{1}{2}x_2^2;\ x_2 \geq 0\}. \tag{65}$$



(a) Isochrones above γ *(b)* Isochrones below γ

Fig. 6. (a) Isochrones for states above the switching curve, (b) states below the switching curve.

The confluence of the locus of points on $\boldsymbol{\gamma}_+$ and $\boldsymbol{\gamma}_-$ is the *switching curve*, depicted in bright orange in Fig. 5, defined as

$$\boldsymbol{\gamma} \triangleq \boldsymbol{\gamma}_+ \cup \boldsymbol{\gamma}_- = \left\{ (x_1, x_2) : x_1 = \frac{1}{2} x_2 |x_2| \right\}. \tag{66}$$

The time-optimal control law, $\boldsymbol{u}^\star$, that solves this problem is unique and is

$$\begin{aligned} \boldsymbol{u}^\star &= \boldsymbol{u}^\star(x_1, x_2) = +1 \quad \forall\, (x_1, x_2) \in \boldsymbol{\gamma}_+ \cup \mathbb{R}_+ \\ \boldsymbol{u}^\star &= \boldsymbol{u}^\star(x_1, x_2) = -1 \quad \forall\, (x_1, x_2) \in \boldsymbol{\gamma}_- \cup \mathbb{R}_- \\ \boldsymbol{u}^\star &= -\text{sgn}\,\{x_2\} \quad \forall\, (x_1, x_2) \in \boldsymbol{\gamma}. \end{aligned} \tag{67}$$

The minimum cost for the problem at hand is the minimum time for states (x_1, x_2) to reach the origin $(0, 0)$, defined as

$$\Phi^\star(\boldsymbol{x}, t) \triangleq t^\star(x_1, x_2) \tag{68}$$

with the associated terminal value

$$-\frac{\partial \Phi^\star(\boldsymbol{x}, t)}{\partial t} = \boldsymbol{H}\left(t, \boldsymbol{x}, \frac{\partial \Phi^\star(\boldsymbol{x}, t)}{\partial t}, \boldsymbol{u} \right) \Big|_{\substack{\boldsymbol{x}=\boldsymbol{x}^\star \\ \boldsymbol{u}=\boldsymbol{u}^\star}} \tag{69}$$

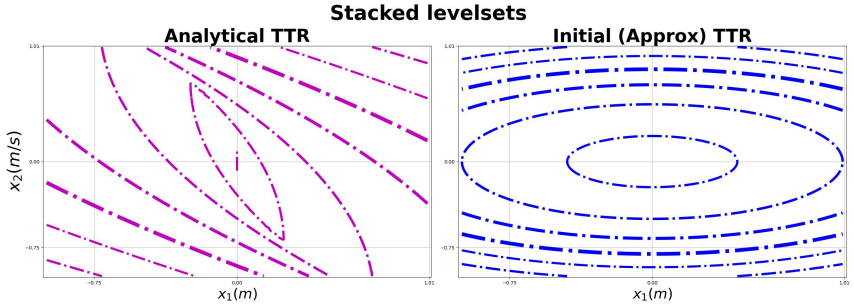
$$\text{with } \boldsymbol{H}(t; \boldsymbol{x}, \boldsymbol{u}, p_1, p_2) = x_2(t) p_1(t) + \boldsymbol{u}(t) p_2(t) \tag{70}$$

and

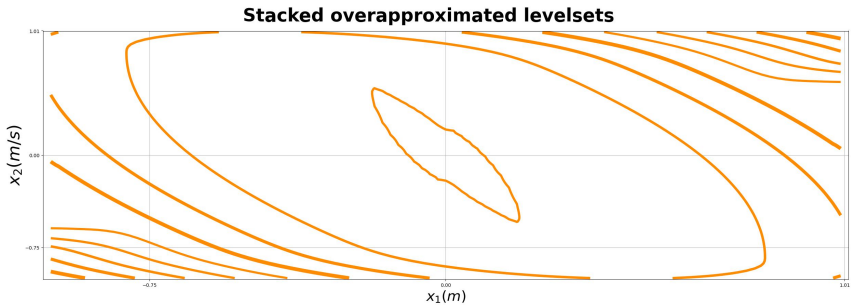
$$p_1 = \frac{\partial t^\star}{\partial x_1}, \;\; p_2 = \frac{\partial t^\star}{\partial x_2}. \tag{71}$$

The HJ equation is given by

$$\begin{aligned} &\frac{\partial \Phi^\star}{\partial t} + x_2 \frac{\partial \Phi^\star}{\partial x_1} - \frac{\partial \Phi^\star}{\partial x_2} = 0 && \text{if } \; x_1 > -\frac{1}{2} x_2 |x_2|, \\ &\frac{\partial \Phi^\star}{\partial t} + x_2 \frac{\partial \Phi^\star}{\partial x_1} + \frac{\partial \Phi^\star}{\partial x_2} = 0 && \text{if } \; x_1 < -\frac{1}{2} x_2 |x_2|, \\ &\frac{\partial \Phi^\star}{\partial t} + x_2 \frac{\partial \Phi^\star}{\partial x_1} - \text{sgn}\{x_2\} \frac{\partial \Phi^\star}{\partial x_2} = 0 && \text{if } \; x_1 = -\frac{1}{2} x_2 |x_2|. \end{aligned} \tag{72}$$



(a) Analytical and Initial BRS (with implicit ellipsoids).



(b) Overapproximated BRS at time $t = 0.25$ secs

Fig. 7. Time to reach the origin at different integration time steps. Top-left: Closed-form Solution to the time to reach the origin problem. Top-right: Implicit representation of the initial TTR solution. Bottom: Lax-Friedrichs Approximation to the TTR the origin problem.

or can be verified to be

$$\frac{2x_2}{\sqrt{4x_1 + 2x_2^2}} - 1 - \frac{2x_2}{\sqrt{4x_1 + 2x_2^2}} \tag{73}$$

The set of states (x_1, x_2) that can be forced to reach the origin in the same minimum time $t^\star$ are the system's *isochrones* which are illustrated in Fig. 6.

A point (x_1, x_2) on the state grid belongs to the set of states $S(t^\star)$ from which it can be forced to the origin $(0, 0)$ in the same minimum time $t^\star$. We call the set $S(t^\star)$ the minimum isochrone. These are the isochrones of the system – akin to the isochrone map used in geography, hydrology, and transportation planning for depicting areas of equal travel time to a goal state. The level sets of (73) correspond to the *isochrones* of the system as illustrated in Fig. 6.

7.2.4 Approximate Time to Reach the Origin. We compare the analytical solution to the *time to reach (TTR) the origin* problem (see Fig. 6) against the approximated TTR solution using a dynamic implicit surface representation of the approximate value function. An ellipsoid with a radius of 1.0 along its major axis was chosen to represent the initial time to reach interface (see Fig. 7a, right inset). We then choose a controller with values ±1 depending on which side of the switch surface Fig. 6 we are on in generating the system's phase portrait illustrated in Fig. 4.

The closed-form solution to the time-to-reach the origin problem on a 2-D grid with x/y axis limits $[[-1, 1], [-1, 1]]$ is shown in the left inset of Fig. 4a. We set out to investigate the result of adding dynamics (with levelsets) to the elliptic implicit representation of the analytical TTR and evaluate the efficacy of our computational scheme. We proceed as shown in Listing 6.

Table 1. Time to Compute BRT.

Experiment	LevPy GPU Time (secs)		LevPy CPU Time (secs)		MATLAB CPU (secs)	
	Global	Avg. local	Global	Avg. local	Global	Avg. local
Rocket Launch						
Double Integrator	14.7657	1.5441				
Air 3D	456.9621	3.14				
Reach Avoid						

```
finite_diff_data = {"innerFunc": termLaxFriedrichs,
  "innerData": {"grid": g, "hamFunc": dint.ham,
  "partialFunc": dint.dissipation,
  "dissFunc": artificialDissipationGLF,
  "CoStateCalc": upwindFirstENO2},
  "positive": False} // direction of approx. growth
```

Listing 5. Overapproximation setup for the double integrator TTR problem.

As a custom, a separate class (see `DoubleIntegrator` in the folder `DynamicalSystems`) holds all the dynamics (cref. equations 59 and 60), switching surface (cref. equations 65, 66, and 67), Hamiltonian (cref. equation 61), dissipation, and costates (cref. 72) of the double integrator plant. Over a twenty-step timespan ranging from 0 to 20, we integrate the right-hand-side of (73) forward in time by the Courant-Friedrichs-Lewy constrained second-order accurate integrator i.e. `odeCFL2` in our library:

```
t, y, ~ = odeCFL2(termRestrictUpdate, t_span, y0, options, finite_diff_data)
```

Listing 6. Overapproximation setup for the double integrator TTR problem.

where `y0` is the initial elliptic function that represents Φ in (73), `options` are the set of integration parameters such as the number of actual timesteps to take in the adaptive integration scheme, the maximum step size and so on. The routine `termRestrictUpdate` restricts the sign of the update of the HJ approximation by either increasing or decreasing the levelset.

A side-by-side comparison of the level sets is shown in Fig. 7a. The approximation to the isochrones by our integration scheme is an overapproximation of the analytical TTR problem. This is illustrated in the right inset of Fig. 7a. Because we are not concerned with the safe set (unlike the example in 7.1), we do not overapproximate the time-to-reach solution. On the overall, we obtain similar isochrones to the analytical result, hence validating our hypothesis.

7.3 Dubins' Game of Two Identical Vehicle

7.4 Computational Time Comparison with LevelSet Toolbox

In this subsection, we will compare the solution for recovering the zero level set of the systems presented in the previous three examples using our library versus Mitchell [22]'s `LevelSet Toolbox` in `MATLAB®`.

The table in 1 depicts the time it takes to process a full global optimization step of the reachable sets/tubes and time-to-reach sets for the various examples we have considered in this work. The columns `Avg. local` depicts the average time it takes per optimization step for computing the relevant set of interest. TO-DO: We see that

8 CONCLUSION

In this paper, we have presented all the essential components of the python version of the levelsets library for numerically resolving HJ PDEs and for advancing co-dimension one interfaces on Cartesian grids. We have motivated the work presented with several numerical examples to demonstrate the efficacy of our library. We encourage users to download the library from the author's github webpage and use it robustly for various problems of interest where speed and scale for the solubility of hyperbolic conservation laws and HJ PDE's are of high importance.

REFERENCES

[1] Michael Athans and Peter L Falb. 2013. *Optimal Control: An Introduction to the Theory and its Applications.* Courier Corporation.

[2] Tamer Basar and Geert Jan Olsder. 1999. *Dynamic Noncooperative Game Theory*. Vol. 23. Siam.

[3] Richard Bellman. 1957. *Dynamic programming*. Princeton University Press.

[4] M. G. Crandall, L. C. Evans, and P. L. Lions. 1984. Some Properties of Viscosity Solutions of Hamilton-Jacobi Equations. *Trans. Amer. Math. Soc.* 282, 2 (1984), 487.

[5] Michael G Crandall and Pierre-Louis Lions. 1983. Viscosity solutions of Hamilton-Jacobi equations. *Transactions of the American mathematical society* 277, 1 (1983), 1–42.

[6] Michael G Crandall and P-L Lions. 1984. Two Approximations of Solutions of Hamilton-Jacobi Equations. *Mathematics of Computation* 43, 167 (1984), 1–19.

[7] Michael G Crandall and Andrew Majda. 1980. Monotone Difference Approximations For Scalar Conservation Laws. *Math. Comp.* 34, 149 (1980), 1–21.

[8] Defense Acquisition University. 2023. Validation. https://www.dau.edu/tools/se-brainbook/Pages/Technical%20Processes/validation.aspx Accessed April 5, 2023.

[9] Defense Acquisition University. 2023. Verification. https://www.dau.edu/tools/se-brainbook/Pages/Technical%20Processes/verification.aspx Accessed April 5, 2023.

[10] Stuart E Dreyfus. 1966. *Control Problems With Linear Dynamics, Quadratic Criterion, and Linear Terminal Constraints*. Technical Report. Rand Corp, Santa Monica Calif.

[11] L.C. Evans and Panagiotis E. Souganidis. 1984. Differential Games And Representation Formulas For Solutions Of Hamilton-Jacobi-Isaacs Equations. *Indiana Univ. Math. J* 33, 5 (1984), 773–797.

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

[13] Lawrence C Evans. 2022. *Partial Differential Equations*. Vol. 19. American Mathematical Society.

[14] Eberhard Hopf. 1950. The Partial Differential Equation $u_t + uu_x = \mu^\star_{xx}$. (1950).

[15] R Isaacs. 1965. Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization. *Kreiger, Huntigton, NY* (1965).

[16] Guang-Shan Jiang and Danping Peng. 2000. Weighted ENO schemes for Hamilton–Jacobi equations. *SIAM Journal on Scientific computing* 21, 6 (2000), 2126–2143.

[17] S.N. Kruzkov. 1970. First Order Quasilinear Equations In Several Independent Variables. *Mathematics of the USSR-Sbornik* (1970).

[18] Pierre-Louis Lions. 1982. *Generalized solutions of Hamilton-Jacobi equations*. Vol. 69. London Pitman.

[19] John Lygeros. 2004. On reachability and minimum cost optimal control. *Automatica* 40, 6 (2004), 917–927.

[20] AW Merz. 1972. The game of two identical cars. *Journal of Optimization Theory and Applications* 9, 5 (1972), 324–343.

[21] Ian Mitchell. 2001. Games of two identical vehicles. *Dept. Aeronautics and Astronautics, Stanford Univ.* July (2001), 1–29.

[22] Ian Mitchell. 2004. A Toolbox of Level Set Methods, version 1.0. *The University of British Columbia, UBC CS TR-2004-09* (July 2004), 1–94.

[23] Ian Mitchell. 2020. A Robust Controlled Backward Reach Tube with (Almost) Analytic Solution for Two Dubins Cars. *EPiC Series in Computing* 74 (2020), 242–258.

[24] Ian M. Mitchell, Alexandre M. Bayen, and Claire J. Tomlin. 2005. A Time-Dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games. *IEEE Trans. Automat. Control* 50, 7 (2005), 947–957.

[25] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. 2017. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*.

[26] S Osher and R Fedkiw. 2004. Level Set Methods and Dynamic Implicit Surfaces. *Applied Mechanics Reviews* 57, 3 (2004), B15–B15.

[27] Stanley Osher and James A. Sethian. 1988. Fronts Propagating with Curvature-Dependent Speed: Algorithms based on Hamilton-Jacobi Formulations. *Journal of Computational Physics* 79, 1 (1988), 12–49.

[28] Stanley Osher and Chi-Wang Shu. 1988. *Efficient Implementation of Essentially Non-oscillatory Shock-capturing Schemes*. Technical Report 2. Hampton, Virginia. 439–471 pages.

[29] Stanley Osher and Chi-Wang Shu. 1991. High-Order Essentially Nonoscillatory Schemes for Hamilton-Jacobi Equations. *SIAM Journal of Numerical Analysis* 28, 4 (1991), 907–922.

[30] James A Sethian. 1996. A Fast Marching Level Set Method For Monotonically Advancing Fronts. *Proceedings of the National Academy of Sciences* 93, 4 (1996), 1591–1595.

[31] James A. Sethian. 2000. Level Set Methods And Fast Marching Methods: Evolving Interfaces In Computational Geometry, Fluid Mechanics, Computer Vision, And Materials Science. *Robotica* 18, 1 (2000), 89–92.

[32] Chi-Wang Shu and Stanley Osher. 1989. Efficient Implementation of Essentially Non-oscillatory Shock-capturing Schemes, II. *Journal of computational physics* 83, 1 (1989), 32–78.
[33] W Murray Wonham. 1985. Linear Multivariable Control: A Geometric Approach. *Applications of Mathematics* 10 (1985).

Received 10 April 2023