The Python LevelSet Toolbox (LevelSetPy)

Lekan Molu
https://github.com/robotsorcerer/levelsetpy

***Abstract*— This paper describes open-source scientific computing contributions in python surrounding the numerical solutions to hyperbolic HJ PDEs viz., their implicit representation on co-dimension one domains; dynamics evolution with levelsets; upwinding spatial derivatives; total variation diminishing Runge-Kutta integration schemes; and their applications to the theory of reachable sets and safety-critical systems. These procedures are increasingly finding interest in multiple research domains including analyzing safety-critical problems in reinforcement learning, robotics and automation; and control engineering among others. We describe a hierarchy of library components, and a representative numerical example included in the online package. Our GPU-accelerated package allows for easy portability to many modern libraries for the numerical analyses of the Bellman and Isaacs equations.**

I. Overview

The *reliability* of the complex algorithms that we design has become paramount given the dangers that may evolve if nominally envisioned system performance falters. Even so, the need for scalable and faster numerical algorithms in software for *verification* and *validation* has become timely given the emerging growth of complexity in the systems that we design and build. The foremost open-source verification software for engineering applications based on Hamilton-Jacobi equations [1, 2] and level set methods [3, 4] is the MATLAB® [5] CPU-based level sets toolbox, before computing via graphical processing units (GPU) became pervasive. Since then, there has been a lot of improvement in computer hardware design, architecture, and code-acceleration.

This paper describes a GPU-accelerated scientific computing software package that is entirely written in python – 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, flow and transport phenomena. We focus on the numerical tools for safety assurance (ascertaining the freedom of a system from harm) in a verification sense in this paper. Accompanying the package are implicit calculus operations on dynamic codimension-one 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 TVD-RK conditioning 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. Finally, extensions to reachability analyses for continuous and hybrid systems, formulated as optimal control or game theory problems using viscosity solutions to HJ PDE's is described. While our emphasis is on the resolution of safe sets in a reachability context for verification settings, the applications of this package extend beyond control engineering applications.

The GPU package, implemented in CuPy [6] and Python, is available on the author's github repository: LevelSetPy. Extensions to other python GPU programming language is straightforward (as detailed in the CuPy interoperability document). The CPU implementation (in Python) can be found at on the cpu-numpy tree of the LevelSetPy repository. In addition, installation instructions are available on the github repository. In all, we try to follow the Python Enhancement Proposals (PEP) 8 style guide[1] as much as possible but in order not to break readability with respect to the original MATLAB®code, we have tried to edge on the side of consistency within the previous project.

II. Background and Motivation

Our interest is in 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 its convection counterpart

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

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 [7]. 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 [7]. The method of "vanishing viscosity", based on the idea of traversing the limit as $\delta \to 0$ in the hyperbolic equation (1)

Microsoft Research, 300 Lafayette Street, NYC
lekanmolu@microsoft.com

[1]Python PEP 8 style guide: peps.python.org/pep-0008/

allows generalized (discontinuous) solutions [8] 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

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

in the class $\text{BUC}(\Omega\times[0,T])\cap C^{2,1}(\Omega\times(0,T])$ i.e. continuous second-order spatial and first order time derivatives for all time $T<\infty$. Crandall and Lions [9] showed that $|\boldsymbol{v}^\delta(\boldsymbol{x},t) - \boldsymbol{v}(\boldsymbol{x},t)| \leq k\sqrt{\delta}$ for $\boldsymbol{x}\in\Omega$ and $t>0$. For most of this paper, we are concerned with *generalized* viscosity solutions of the manner described by (3).

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 [10], the Hamiltonian is related to the *backward* reachable set [11] of a dynamical system. Mitchell [12] 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 [8, 13] constitutes the safe set of a reachability problem [10]. We refer interested readers on the technicalities of the theory to [12]'s paper.

The well-known `LevelSet Toolbox` [5] is the consolidated `MATLAB®` package that contains the grid 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 [6] and present representative numerical examples. Our **contributions** are as follows:

1) we describe the details of the LevelSetPy package, starting with the common implicit surfaces that are used as initial conditions to represent $\boldsymbol{v}(\boldsymbol{x},t)$;
2) we then describe the upwinding spatial derivative, 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 then conclude with a representative example, namely, the barrier surface for two adversarial rockets traveling on a plane. More examples abound on the online code repository.

The rest of this paper is structured as follows. We describe the geometry of (and Boolean operations on) implicit function representations of continuous-time value functions described by (1) using Cartesian grids in section III. Spatial derivatives to scalar conservation laws are discussed in section IV, and temporal discretization schemes for these conservation laws follow thereafter. In section VI we formulate a didactic two-rockets game and show how to define the numerical safe backward reachable tubes amenable to HJ PDEs within a geometrical verification framework. We conclude the paper in section VII. Additional `python` examples, `jupyter` notebooks, and representative problems are provided in the online package.

III. The LevelSetPy Python Package

Let us now describe how solutions to the value function in (1) and (3) are constructed in our software package. 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.

A. Geometry of Implicit Surfaces and Layouts

As stated before, solutions to the value function (1) are implicitly represented on co-dimension one surfaces in $\mathbb{R}^n$. We discuss implicit surface functions' contruction in LevelSetPy, CPU memory, and GPU transfers.

The implicit interfaces are typically isocontours of some function, $f(x)$. Implicit construction is attractive as it requires less number of points to construct a function than explicit forms. The zero isocontour (or the zero levelset) of a reachable optimal control problem is equivalent to the safety set or backward reachable tube [12]; and for a differential game, it is the *barrier surface* between the *capture* and *escape zones* for all trajectories that emanate from a system.

B. 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.

```
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.

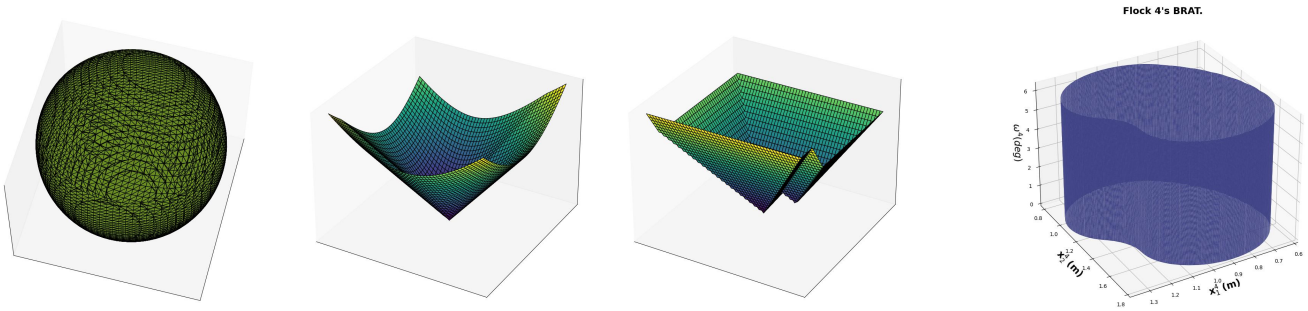

Fig. 1: Zero level set of **(a)** a sphere on a 3D grid; **(b)** union of two 3D spheres implicitly constructed on a 2D grid; **(c)** the union of rectangles on a 2D grid; **(d)** the union of multiple cylinders on a 3D grid.

A grid `data structure`, **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 $\mathbf{g}.xs$ as a list across all the dimensions of the grid; (iii) grid dimension $\mathbf{g}.dim$, denoting the number of Cartesian axes needed for representing the state x[2]; (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).

C. Implicit Surface Representations: Levelsets

We treat coordinates as functional arguments using a fixed level set of continuous function $\boldsymbol{v} : \mathbb{R}^n \to \mathbb{R}$. We use signed distance functions to represent the dynamics 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 project page.

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, where the 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.

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

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

D. Calculus on Implicit Function Representations

Geometrical operations on implicitly defined functions carries through in the package as follows. Let $\boldsymbol{v}_1(x)$ and $\boldsymbol{v}_2(x)$ be two signed distance representations, then the union of the interior of both is simply $\min(\boldsymbol{v}_1(x), \boldsymbol{v}_2(x))$ (illustrations in Fig. 1 b and c and d). 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`.

IV. Spatial Discretization: Upwinding

In this section, we discuss higher-order upwinding schemes that mimic high-order essentially non-socillatory (ENO) [14] schemes for computing the spatial derivatives $\boldsymbol{v}_x$ for the numerical viscosity solutions to levelset PDE's of the *Eulerian form* (introduced in (4)). Codebases for procedures herewith described are in the folder `SpatialDerivatives`. Using the Eulerian form of the levelset equation,

$$\boldsymbol{v}_t + \boldsymbol{F} \cdot \nabla \boldsymbol{v} = 0 \tag{4}$$

where $\boldsymbol{F}$ is the speed function, the implicit function representation v_t (see §III) is used both to denote and 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 (4) the evolution of the implicit function on the zero levelset yields the Eulerian form

$$\boldsymbol{v}_t + f_x \boldsymbol{v}_x + f_y \boldsymbol{v}_y + f_z \boldsymbol{v}_z = 0 \tag{5}$$

of the interface evolution given that the interface encapsulates the implicit representation $\boldsymbol{v}$. In our implementation, we define $\boldsymbol{v}$ throughout the computational domain Ω.

Let us first construct the general form of a spatial upwinding scheme i.e.

$$D^- \boldsymbol{v} = \frac{\partial \boldsymbol{v}}{\partial x} \approx \frac{\boldsymbol{v}_{i+1} - \boldsymbol{v}_i}{\Delta x}, D^+ \boldsymbol{v} \approx \frac{\boldsymbol{v}_i - \boldsymbol{v}_{i-1}}{\Delta x}, \tag{6}$$

where $\boldsymbol{v}$ 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 (5) becomes $(1/\Delta t) \cdot \boldsymbol{v}^{n+1} - \boldsymbol{v}^n + f_x^n \boldsymbol{v}_x^n + f_y^n \boldsymbol{v}_y^n + f_z^n \boldsymbol{v}_z^n = 0$.

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

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

$$\frac{\boldsymbol{v}_i^{n+1} - \boldsymbol{v}_i^n}{\Delta t} + f_i^n (\boldsymbol{v}_x)_i^n = 0 \tag{7}$$

where $(\boldsymbol{v}_x)_i$ denotes the spatial derivative of $\boldsymbol{v}$ w.r.t x at the point i. We now discuss the specific implementations of the upwinding schemes in our library.

A. First-order accurate upwinding discretization

If $f_i > 0$ in (7), 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 [4, §3.1]) and vice versa if $f_i < 0$. We therefore follow the standard upwinding method by using (6): 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 [5] and in our `SpatialDerivatives` folder, we name this function `upwindFirstFirst`.

B. ENO Polynomial Interpolation of Solutions

Using a divided differencing table, essentially non-oscillatory (ENO) polynomial interpolation of the discretization [14] 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 $D_i^0\boldsymbol{v} = \boldsymbol{v}_i$, and the first, second, and third order divided differences of $\boldsymbol{v}$ as the midway between grid nodes i.e.

$$D^1_{i+1/2}\boldsymbol{v} = \frac{D^0_{i+1}\boldsymbol{v} - D^0_i\boldsymbol{v}}{\Delta x},\ D^2_i\boldsymbol{v} = \frac{D^1_{i+1/2}\boldsymbol{v} - D^1_{i-1/2}\boldsymbol{v}}{2\Delta x}, \tag{8a}$$

$$D^3_{i+1/2}\boldsymbol{v} = \frac{D^2_{i+1}\boldsymbol{v} - D^2_i\boldsymbol{v}}{3\Delta x}. \tag{8b}$$

Then, an essentially non-oscillating polynomial of the form

$$\boldsymbol{v}(x) = Q_0(x) + Q_1(x)!Q_2(x) + Q_3(x) \tag{9}$$

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.

$$\boldsymbol{v}_x(x_i) = Q_1'(x_i) + Q_2'(x_i) + Q_3'(x_i). \tag{10}$$

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

$$Q_1(x) = (D^1_{k+1/2}\boldsymbol{v})(x - x_i),\ Q_1'(x_i) = D^1_{k+1/2}\boldsymbol{v}. \tag{11}$$

We follow [4]'s recommendation in avoiding interpolating near large oscillations in gradients. Therefore, we choose a constant c such that

$$c = \begin{cases} D^2_k\boldsymbol{v} & \text{if } |D^2_k\boldsymbol{v}| \le |D^2_{k+1}\boldsymbol{v}| \\ D^2_{k+1}\boldsymbol{v} & \text{otherwise} \end{cases} \tag{12}$$

so that $Q_2(x) = c(x - x_k)(x - x_{k+1})$, $Q_2'(x_i) = c(2i - 2k - 1)\Delta x$ 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

$$c^\star = \begin{cases} D^3_{k^\star+1/2} & \text{if } |D^3_{k^\star+1/2}\boldsymbol{v}| \le |D^3_{k^\star+3/2}\boldsymbol{v}| \\ D^3_{k^\star+3/2}\boldsymbol{v} & \text{if } |D^3_{k^\star+1/2}\boldsymbol{v}| > |D^3_{k^\star+3/2}\boldsymbol{v}|. \end{cases} \tag{13}$$

Whence, we have

$$Q_3(x) = c^\star(x - x_{k^\star})(x - x_{k^\star} + 1)(x - x_{k^\star} + 2) \tag{14a}$$

$$Q_3'(x_i) = c^\star(3(i - k^\star)^2 - 6(i - k^\star) + 2)(\Delta x)^2 \tag{14b}$$

for the third-order accurate correction to the approximated upwinding scheme (9). This is implemented as a routine in `upwindFirstENO3aHelper` and called as `upwindFirstENO3` in the `SpatialDerivatives` folder.

C. 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 [15]. 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 $\{i-1, \cdots, i+3\}$ 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

$$\boldsymbol{v}^{-,0}_{x,i} = \frac{1}{3}D^+\boldsymbol{v}_{i-3} - \frac{7}{6}D^+\boldsymbol{v}_{i-2} + \frac{11}{6}D^+\boldsymbol{v}_{i-1} \tag{15a}$$

$$\boldsymbol{v}^{-,1}_{x,i} = -\frac{1}{6}D^+\boldsymbol{v}_{i-2} + \frac{5}{6}D^+\boldsymbol{v}_{i-1} + \frac{1}{3}D^+\boldsymbol{v}_i \tag{15b}$$

$$\boldsymbol{v}^{-,2}_{x,i} = -\frac{1}{3}D^+\boldsymbol{v}_{i-1} + \frac{5}{6}D^+\boldsymbol{v}_i - \frac{1}{6}D^+\boldsymbol{v}_{i+1} \tag{15c}$$

where $\boldsymbol{v}^{-,p}_{x,i}$ 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

$$\boldsymbol{v}^-_{x,i} = w_0\boldsymbol{v}^{-,0}_{x,i} + w_1\boldsymbol{v}^{-,1}_{x,i} + w_2\boldsymbol{v}^{-,2}_{x,i}. \tag{16}$$

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 $\boldsymbol{v}^-_{x,i}$

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

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 (15) can be estimated as recommended in [4, §3.4] so that if

$$\alpha_1 = 0.1/(\sigma_1 + \epsilon)^2,\ \alpha_2 = 0.6/(\sigma_2 + \epsilon)^2,\ \alpha_3 = 0.1/(\sigma_3 + \epsilon)^2 \tag{17}$$

for

$$\sigma_1 = \frac{13}{12}(D^+\boldsymbol{v}_{i-3} - 2D^+\boldsymbol{v}_{i-2} + D^+\boldsymbol{v}_{i-1})^2 + \frac{1}{4}(D^+\boldsymbol{v}_{i-3} - 4D^+\boldsymbol{v}_{i-2} + 3D^+\boldsymbol{v}_{i-1})^2, \tag{18a}$$

$$\sigma_2 = \frac{13}{12}(D^+\boldsymbol{v}_{i-2} - 2D^+\boldsymbol{v}_{i-3} + D^+\boldsymbol{v}_i)^2 + \frac{1}{4}(D^+\boldsymbol{v}_{i-2} - D^+\boldsymbol{v}_i)^2, \tag{18b}$$

$$\sigma_3 = \frac{13}{12}(D^+\boldsymbol{v}_{i-1} - 2D^+\boldsymbol{v}_i + D^+\boldsymbol{v}_{i+1})^2 + \frac{1}{4}(3D^+\boldsymbol{v}_{i-1} - 4D^+\boldsymbol{v}_i + D^+\boldsymbol{v}_{i+1})^2, \tag{18c}$$

and

$$\epsilon = 10^{-6}\max\{D^+\boldsymbol{v}_{i-3}, D^+\boldsymbol{v}_{i-2}, D^+\boldsymbol{v}_{i-1}, D^+\boldsymbol{v}_i D^+\boldsymbol{v}_{i+1}\} + 10^{-99} \tag{19}$$

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

$$w_1 = \alpha_1/\sum_{i=1}^{3}\alpha_i,\ w_2 = \alpha_2/\sum_{i=1}^{3}\alpha_i,\ w_3 = \alpha_3/\sum_{i=1}^{3}\alpha_i. \tag{20}$$

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`.

D. Lax-Friedrichs Monotone Difference Schemes

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

$$\boldsymbol{v}_t + \sum_{i=1}^{N} f_i(\boldsymbol{v})_{x_i} = 0 \text{ for } t > 0, \boldsymbol{x} = (x_1, \cdots, x_N) \in \mathbb{R}^n$$
$$\boldsymbol{v}(\boldsymbol{x}, 0) = \boldsymbol{v}_0(\boldsymbol{x}), \text{ for } \boldsymbol{x} \in \mathbb{R}^n \tag{21}$$

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 [16]

$$\boldsymbol{v}^{n+1}_j = \boldsymbol{v}^n_j - \frac{\lambda_x}{2}\Delta^0_x f(\boldsymbol{v}^n_j) + \frac{1}{2}\Delta^+_x\Delta^-_x\boldsymbol{v}^n_j. \tag{22}$$

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

$$g(\boldsymbol{v}_j, \boldsymbol{v}_{j-1}) = \frac{f(\boldsymbol{v}_j) + f(\boldsymbol{v}_{j-1})}{2} - \frac{1}{2}\lambda_x(\boldsymbol{v}_j - \boldsymbol{v}_{j-1}), \tag{23}$$

we may write $\boldsymbol{v}^{n+1}_j = \boldsymbol{v}^n_j - \lambda^+_x(\boldsymbol{v}_j, \boldsymbol{v}_{j-1})$.

The Lax-Friedrichs scheme is monotone on the interval $[a, b]$ if the CFL condition $\lambda_x \max_{a\le\boldsymbol{v}\le b}|f'(\boldsymbol{v})| \le 1$ for $(a, b) > 0$ and the upwind differencing scheme for a nondecreasing f is $\boldsymbol{v}^{n+1}_j = \boldsymbol{v}^n_j - \lambda_x\Delta^+_x f(\boldsymbol{v}^n_{j-1})$. For a non-increasing f, we have $\boldsymbol{v}^{n+1}_j = \boldsymbol{v}^n_j - \lambda_x\Delta^+_x f(\boldsymbol{v}^n_j)$. Our Lax-Friedrichs implementation is implemented in the `ExplicitIntegration/Term` folder.

V. 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 IV. We adopt the *method of lines* (MOL) used in converting the time-dependent PDEs to ODEs. Our presentation follows the total variation diminishing (TVD) Runge Kutta (RK) schemes with Courant-Friedrichs-Lewy (CFL) conditioning imposed for stability as presented in [17] and implemented in `MATLAB®` in [5].

A. Higher-Order TVD-RK Time Discretizations

To adopt the method of lines, the N-dimensional levelset representation of $\boldsymbol{v}$ is first rolled into a 1-D vector and an adaptive integration step size, Δt, is chosen to guarantee stability following the recommendation in [18]. The forward Euler algorithm thus becomes

$$\boldsymbol{v}(x, t + \Delta t) = \boldsymbol{v}(x, t) + \Delta t\Upsilon(x, \boldsymbol{v}(x, t)) \tag{24}$$

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 [5]'s code layout to provide consistency for MATLAB users). 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. The routine `schemeFunc` is typically one of the Lax-Friedrichs approximation routines (implemented as `termLaxFriedrichs`) in the folder `ExplicitIntegration/Term`. It approximates the HJ equation based on dissipation functions (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 $t^n + \Delta t$,

$$\frac{\boldsymbol{v}^{n+1} - \boldsymbol{v}^n}{\Delta t} + F^n \cdot \nabla\boldsymbol{v}^n = 0. \tag{25}$$

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

$$\frac{\boldsymbol{v}^{n+2} - \boldsymbol{v}^{n+1}}{\Delta t} + F^{n+1} \cdot \nabla \boldsymbol{v}^{n+1} = 0 \tag{26}$$

before a convex combination of the initial value function and the result of the preceding Euler steps is taken in the following averaging step, $\boldsymbol{v}^{n+1} = \frac{1}{2}\{\boldsymbol{v}^n + \boldsymbol{v}^{n+2}\}$. The equation in the foregoing produces the second-order accurate TVD approximation to $\boldsymbol{v}$ 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, $\boldsymbol{v}^{n+1/2} = \frac{1}{4}\{3\boldsymbol{v}^n + \boldsymbol{v}^{n+2}\}$ 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

$$\frac{\boldsymbol{v}^{n+\frac{3}{2}} - \boldsymbol{v}^{n+\frac{1}{2}}}{\Delta t} + F^{n+\frac{1}{2}} \cdot \nabla \boldsymbol{v}^{n+\frac{1}{2}} = 0 \tag{27}$$

together with the averaging scheme, $\boldsymbol{v}^{n+1} = \frac{1}{3}\{\boldsymbol{v}^n + 2\boldsymbol{v}^{n+\frac{3}{2}}\}$ to produce a third-order accurate approximation to $\boldsymbol{v}$ at time $t^n + \Delta t$, implemented as `odeCFL3`.

VI. Numerical Validation

In this section, we will a representative problem and amend it to HJ PDE forms where numerical solutions can be resolved with our `LevelSetPy` toolbox.The library has been tested on numerous problems including `transport`, `differential games`, and `time-to-reach` problem classes which are all available in the "Examples" folder; however, for the sake of brevity we will only report one result here on a differential game here.

We consider a *differential game*. In our setting, we do not necessarily analyze a single game, but rather a *collection/-family of games*, $\Upsilon = \{\Gamma_1, \cdots, \Gamma_g\}$. Each game within a differential game may be characterized as a pursuit-evasion *game*, Γ. 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 ($\boldsymbol{E}$).

To address our desiderata, we must settle upon how best should $\boldsymbol{P}$ pursue $\boldsymbol{E}$. 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. The task is to assay the *game of kind* [13] for the envelope of the capturable states under which capture can occur. This introduces the *barrier* hypersurface (or the backward reachable tube [12]) which separates, in the initial conditions space, the hypersurface of capture from those of escape. Here, optimal strategies are not unique, but rather are a *legion*.

A. The Rockets Launch Problem

We consider the rocket launch problem of Dreyfus [19] and amend it to a differential game between two identical rockets, $\boldsymbol{P}$ and $\boldsymbol{E}$, on an (x, z) cross-section of a Cartesian plane. We want to compute the backward reachable tube (BRT) of the *approximate* terminal surface's boundary for a predefined target set over a time horizon (i.e. the target tube). The BRT entails the state-space regions for which the min-max operation over either strategy of $\boldsymbol{P}$ and $\boldsymbol{E}$ is below 0. This BRT constitutes where the variational HJI PDE is exactly zero.

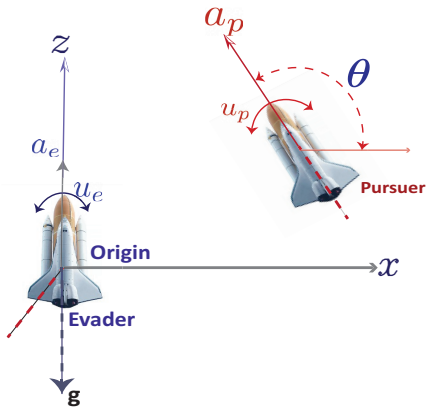

Fig. 2: Motion of two rockets on a Cartesian $\boldsymbol{xz}$-plane with a thrust inclination in relative coordinates given by $\theta := u_p - u_e$.

For a two-player differential game, let $\boldsymbol{P}$ and $\boldsymbol{E}$ share identical dynamics in a general sense. The coordinates of $\boldsymbol{P}$ are freely chosen; however, $\boldsymbol{E}$'s are a distance $\boldsymbol{\phi}$ away from (x, z) at plane's origin (see Fig. 2) so that the $\boldsymbol{PE}$ vector's inclination measured counterclockwise from the $\boldsymbol{x}$ axis is θ.

Being a free endpoint problem, let the states of $\boldsymbol{P}$ and $\boldsymbol{E}$ be denoted by $(\boldsymbol{x}_p, \boldsymbol{x}_e)$. Furthermore, let the rockets be driven by their thrusts, denoted by (u_p, u_e) for $\boldsymbol{P}$ and $\boldsymbol{E}$ respectively (see Figure 2). Fix the range of the rockets so that what is left of the motion of either $\boldsymbol{P}$ or $\boldsymbol{E}$'s is restricted to orientation on the (x, z) plane as illustrated in Fig. 2. The relevant *kinematic equations* (KE) are derived off [19]'s single rocket launch as

$$\dot{x}_{2e} = x_{4e}; \qquad \dot{x}_{2p} = x_{4p}, \tag{28a}$$

$$\dot{x}_{4e} = a \sin u_e - g; \qquad \dot{x}_{4p} = a \sin u_p - g \tag{28b}$$

where a and g are respectively the acceleration and gravitational accelerations (in feet per square second) [3].

Our desideratum is determining if capture can be achieved at all in a "yes-or-no" fashion. Therefore, we pose the game over a finite range over outcomes so that the game at hand assumes Isaac's [13] description of *a game of kind*. $\boldsymbol{P}$ can achieve as much proximity to a given *target set* as much as possible while $\boldsymbol{E}$ is set up to protect the *target set*. For example, one may take $\boldsymbol{P}$ as seeking to penetrate a (closed) territory (called *target*) under guard by player $\boldsymbol{E}$; and $\boldsymbol{P}$'s goal may be to maximize the time of play so as to penetrate the barrier surface of the target. $\boldsymbol{E}$ seeks to protect a given target's surface. As long as $\boldsymbol{E}$ remains within this *backward*

[3] We set $a = 1 ft/sec^2$ and $g = 32 ft/sec^2$ in our simulation.

reachable tube (or BRT), $\boldsymbol{P}$ cannot cause damage or exercise an action of deleterious consequence on, say, the territory being guarded by $\boldsymbol{E}$.

Setting up $\boldsymbol{E}$ to maximize a payoff quantity with the largest possible margin or at least frustrate the efforts of $\boldsymbol{P}$ with minimal collateral damage while the pursuer minimizes this 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 problem parameters.

$\boldsymbol{P}$'s motion relative to $\boldsymbol{E}$'s along the $(\boldsymbol{x}, \boldsymbol{z})$ plane includes the relative orientation shown in Fig. 2 as $\theta = u_p - u_e$ – the control input. Following the conventions in Fig. 2, the game's relative equations of motion in *reduced space* [13, §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

$$\dot{\boldsymbol{x}} = \begin{cases} \dot{x} &= a_p \cos\theta + u_e x, \\ \dot{z} &= a_p \sin\theta + a_e + u_e x - g, \\ \dot{\theta} &= u_p - u_e. \end{cases} \tag{29}$$

The capture radius of the origin-centered circle $\boldsymbol{\phi}$ (where $\boldsymbol{\phi} = 1.5$ ft) is $\|\boldsymbol{PE}\|_2$ so that $\boldsymbol{\phi}^2 = x^2 + z^2$. All capture points are specified by the variational HJ PDE [12]:

$$\frac{\partial \boldsymbol{\phi}}{\partial t}(\boldsymbol{x}, t) + \min\left[0, \boldsymbol{H}(\boldsymbol{x}, \frac{\partial \boldsymbol{\phi}(\boldsymbol{x}, t)}{\partial \boldsymbol{x}})\right] \leq 0, \tag{30}$$

with Hamiltonian given by

$$\begin{aligned} \boldsymbol{H}(\boldsymbol{x}, p) = - \max_{u_e \in [\underline{u}_e, \bar{u}_e]} \min_{u_p \in [\underline{u}_p, \bar{u}_p].} & \begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} \\ & \begin{bmatrix} a_p \cos\theta + u_e x \\ a_p \sin\theta + a_e + u_p x - g \\ u_p - u_e \end{bmatrix}. \end{aligned} \tag{31}$$

Here, the co-states p is defined with a strict corresponding property, and $[\underline{u}_e, \bar{u}_e]$ denotes the extremals that the evader must choose as input in response to the extremal controls that the pursuer plays i.e. $[\underline{u}_p, \bar{u}_p]$. Rather than resort to analytical *geometric reasoning*, we may analyze possibilities of behavior by either agent via a principled numerical simulation. This is what we present next. From (31), set $\underline{u}_e = \underline{u}_p = \underline{u} \triangleq -1$ and $\bar{u}_p = \bar{u}_e = \bar{u} \triangleq +1$ so that $\boldsymbol{H}(\boldsymbol{x}, p)$ is

$$\begin{aligned} & - \max_{u_e \in [\underline{u}_e, \bar{u}_e]} \min_{u_p \in [\underline{u}_p, \bar{u}_p]} \begin{bmatrix} p_1(a_p \cos\theta + u_e x) + \\ p_2(a_p \sin\theta + a_e + \\ u_p x - g) + p_3(u_p - u_e) \end{bmatrix}, \\ & \triangleq -a p_1 \cos\theta - p_2(g - a - a\sin\theta) - \bar{u}|p_1 x + p_3| \\ & \quad + \underline{u}|p_2 x + p_3|, \end{aligned} \tag{32}$$

where the last line follows from setting $a_e = a_p \triangleq a$.

For the target set guarded by $\boldsymbol{E}$, we choose an implicitly constructed cylindrical mesh on a three-dimensional grid. The grid's nodes are uniformly spaced apart at a resolution of 100 points per dimension over the interval $[-64, 64]$. In numerically solving for the Hamiltonian (32), a TVD-RK discretization scheme [14] based on fluxes is used in choosing smooth nonoscillatory results as described in §V. Denote by (x, y, z) a generic point in $\mathbb{R}^3$ so that given mesh sizes Δx, Δy, Δz, $\Delta t > 0$, letters u, v, w represent functions on the x, y, z lattice: $\Delta = \{(x_i, y_j, z_k) : i, j, k \in \mathbb{Z}\}$. are the dissipation coefficients, controlling the level of numerical viscosity in order to realize a stable solution that is physically realistic [9]. Here, the subscripts of $\boldsymbol{H}$ are the partial derivatives w.r.t the subscript variable.

```
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 non-oscillatory 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 $\boldsymbol{v}(\boldsymbol{x}, t)$. The monotone spatial upwinding scheme used (here `termLaxFriedrichs` described in §IV-D) is passed into the `innerFunc` query field. The explicit form of the Hamiltonian (see (32)) 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 that we intend to overapproximate the value function, we specify a `True` parameter for the `positive` query field.

Safety is engendered by having the evader respond optimally to the pursuer at various times during the game. We are thus interested in the entire safety set over the time interval of play (i.e. the safety tube). The backward reachable tube (BRT) [12], under the control strategies of $\boldsymbol{P}$ or $\boldsymbol{E}$, is a part of the phase space that constitutes $\Omega \times T$. 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 [12]). 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 [10, 21] thus affords us an ability to numerically reason about the behavior of these two rockets

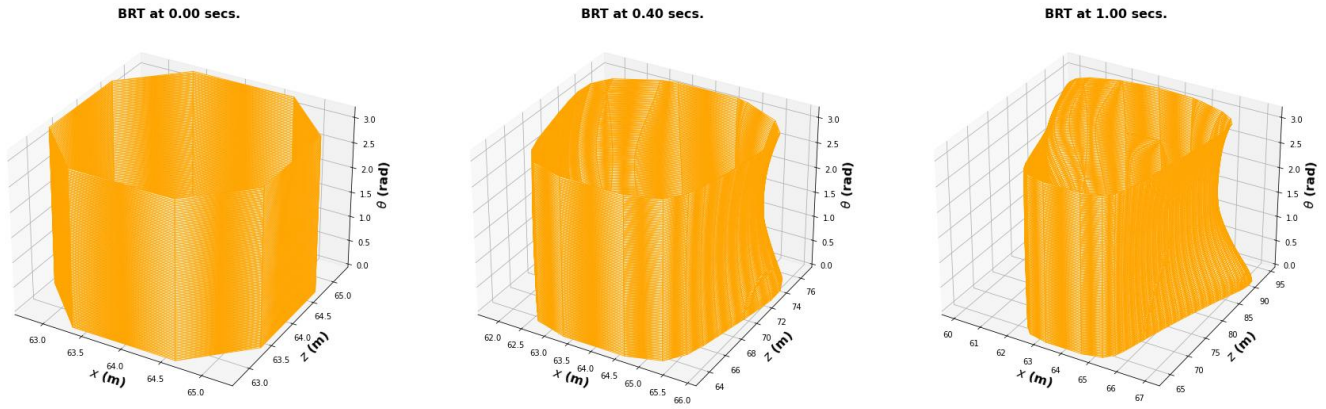

Fig. 3: *(Left to Right)*: Backward reachable tubes (capture surfaces) for the rocket system (cf. Fig. 2) optimized for the paths of slowest-quickest descent in equation (31) at various time steps during the differential game. In all, the BRTs were computed using the method outlined in [4, 11, 20]. We set $a_e = a_p = 1ft/sec^2$ and $g = 32ft/sec^2$ as in Dreyfus' original example.

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 IV and the corresponding codes described).

VII. Conclusion

HJ PDE's are increasingly becoming a useful tool in control and learning applications. We have presented all the essential components of the python version of the `LevelSetPy` library for numerically resolving HJ PDEs and for advancing co-dimension one interfaces on Cartesian grids. We have motivated the work presented with a numerical example to demonstrate the efficacy of our numerical implementation. We remark that further examples are documented in our online library for users who may wish to utilize our toolbox. Further analysis (indicated on the online package repository) delineates the CPU and GPU capabilities of the library and the average time to compute solutions across diverse tasks.

References

[1] S. Kruzkov, "First Order Quasilinear Equations In Several Independent Variables," *Mathematics of the USSR-Sbornik*, 1970. 1

[2] L. C. Evans, *Partial Differential Equations*. American Mathematical Society, 2022, vol. 19. 1

[3] J. A. Sethian, "Level Set Methods And Fast Marching Methods: Evolving Interfaces In Computational Geometry, Fluid Mechanics, Computer Vision, And Materials Science," *Robotica*, vol. 18, no. 1, pp. 89–92, 2000. 1

[4] S. Osher and R. Fedkiw, "Level Set Methods and Dynamic Implicit Surfaces," *Applied Mechanics Reviews*, vol. 57, no. 3, pp. B15–B15, 2004. 1, 4, 5, 8

[5] I. Mitchell, "A toolbox of level set methods, version 1.0," *The University of British Columbia, UBC CS TR-2004-09*, pp. 1–94, July 2004. 1, 2, 4, 5

[6] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "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)*, 2017. 1, 2

[7] M. G. Crandall and P.-L. Lions, "Viscosity solutions of hamilton-jacobi equations," *Transactions of the American mathematical society*, vol. 277, no. 1, pp. 1–42, 1983. 1

[8] L. Evans and P. E. Souganidis, "Differential Games And Representation Formulas For Solutions Of Hamilton-Jacobi-Isaacs Equations," *Indiana Univ. Math. J*, vol. 33, no. 5, pp. 773–797, 1984. 2

[9] M. G. Crandall and P.-L. Lions, "Two Approximations of Solutions of Hamilton-Jacobi Equations," *Mathematics of Computation*, vol. 43, no. 167, pp. 1–19, 1984. 2, 7

[10] J. Lygeros, "On reachability and minimum cost optimal control," *Automatica*, vol. 40, no. 6, pp. 917–927, 2004. 2, 7

[11] I. Mitchell, "A Robust Controlled Backward Reach Tube with (Almost) Analytic Solution for Two Dubins Cars," *EPiC Series in Computing*, vol. 74, pp. 242–258, 2020. 2, 8

[12] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A Time-Dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005. 2, 6, 7

[13] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization.* Kreiger, Huntigton, NY, 1999. 2, 6, 7

[14] S. Osher and C.-W. Shu, "High-Order Essentially Nonoscillatory Schemes for Hamilton-Jacobi Equations," *SIAM Journal of Numerical Analysis*, vol. 28, no. 4, pp. 907–922, 1991. 3, 4, 7

[15] G.-S. Jiang and D. Peng, "Weighted eno schemes for hamilton–jacobi equations," *SIAM Journal on Scientific computing*, vol. 21, no. 6, pp. 2126–2143, 2000. 4

[16] M. G. Crandall and A. Majda, "Monotone Difference Approximations For Scalar Conservation Laws," *Mathematics of Computation*, vol. 34, no. 149, pp. 1–21, 1980. 5

[17] S. Osher and C.-W. Shu, "Efficient Implementation of Essentially Non-oscillatory Shock-capturing Schemes," Hampton, Virginia, Tech. Rep. 2, 1988. 5

[18] C.-W. Shu and S. Osher, "Efficient Implementation of Essentially Non-oscillatory Shock-capturing Schemes, II," *Journal of computational physics*, vol. 83, no. 1, pp. 32–78, 1989. 5

[19] S. E. Dreyfus, "Control Problems With Linear Dynamics, Quadratic Criterion, and Linear Terminal Constraints," Rand Corp, Santa Monica Calif, Tech. Rep., 1966. 6

[20] M. G. Crandall, L. C. Evans, and P. L. Lions, "Some Properties of Viscosity Solutions of Hamilton-Jacobi Equations," *Transactions of the American Mathematical Society*, vol. 282, no. 2, p. 487, 1984. 8

[21] I. Mitchell, "Games of two identical vehicles," *Dept. Aeronautics and Astronautics, Stanford Univ.*, no. July, pp. 1–29, 2001. 7