Differentiable Physics Simulations with Contacts: Do They Have Correct Gradients w.r.t. Position, Velocity and Control?

Yaofeng Desmond Zhong [1] **Jiequn Han** [2] **Georgia Olympia Brikis** [1]

Abstract

In recent years, an increasing amount of work has focused on differentiable physics simulation and has produced a set of open source projects such as Tiny Differentiable Simulator, Nimble Physics, diffTaichi, Brax, Warp, Dojo and DiffCoSim. By making physics simulations end-to-end differentiable, we can perform gradient-based optimization and learning tasks. A majority of differentiable simulators consider collisions and contacts between objects, but they use different contact models for differentiability. In this paper, we overview four kinds of differentiable contact formulations - linear complementarity problems (LCP), convex optimization models, compliant models and position-based dynamics (PBD). We analyze and compare the gradients calculated by these models and show that the gradients are not always correct. We also demonstrate their ability to learn an optimal control strategy by comparing the learned strategies with the optimal strategy in an analytical form. The codebase to reproduce the experiment results is available at https://github.com/DesmondZhong/diff_sim_grads.

1. Introduction

With rapid advances and development of machine learning and automatic differentiation tools, a family of techniques emerge to make physics simulation end-to-end differentiable (Liang & Lin, 2020). These differentiable physics simulators make it easy to use gradient-based methods for learning and control tasks, such as system identification (Zhong et al., 2021; Le Lidec et al., 2021; Song & Boularias, 2020a), learning to slide unknown objects (Song & Boularias, 2020b) and shape optimization (Strecke & Stueckler, 2021; Xu et al., 2021). These applications demonstrate the potential of differentiable simulations in solving control and design problems that are hard to solve by traditional tools. Compared to black-box neural networks counterparts, differentiable simulations utilize physical models to provide more reliable gradient information and better interpretability, which is beneficial to various learning tasks involving physics simulations.

A crucial challenge of making physics simulation differentiable is the non-smoothness of contact events. In the literature, different techniques have been proposed to compute gradients in dynamics involving contact events. A detailed comparison of these techniques is necessary for researchers to understand their pros and cons.

In this paper, we first overview four kinds of differentiable contact formulations - linear complementarity problems (LCP), convex optimization models, compliant models and position-based dynamics (PBD). Even though differentiable simulation and contact models has been studied for deformable objects (Rojas et al., 2021; Qiao et al., 2021a; Du et al., 2021a) and cloth (Liang et al., 2019; Li et al., 2021), we focus on collisions between rigid bodies in our benchmark experiments. We seek to answer a simple yet important question - do these differentiable contact formulations compute the correct gradients w.r.t. position, velocity and control? We implement different types of differentiable simulations on two examples where the analytical gradients can be derived in a closed form. By comparing the gradients computed by simulations with analytical gradients, we observe that not all the computed gradients are correct and the gradients computed by different open source implementations do not agree. Our results reveal the limitation of current differentiable simulators and open up new research opportunities to develop more reliable physics simulators.

2. Differentiable Physics Simulation with Contacts

In this section, we overview different kinds of differentiable contact models and discuss how the gradients through the contact events are derived and calculated. We start with linear complementarity problems and convex optimization models, both of which treat contact events as instantaneous

[1]Siemens Technology [2]Flatiron Institute. Correspondence to: Y. D. Zhong <yaofeng.zhong@siemens.com>.

2nd AI4Science Workshop at the 39th International Conference on Machine Learning (ICML), 2022. Copyright 2022 by the author(s).

velocity changes during simulation. The goal of these two families of methods is to solve velocity impulses.

2.1. Linear Complementarity Problems

Solving velocity impulses in a frictional contact event can be formulated as a nonlinear complementarity problem (NCP), where the friction cone constraint is nonlinear. A recent differentiable simulator Dojo (Howell et al., 2022) designs a customized solver to solve the NCP problem, and leverages the implicit-function theorem to derive the gradients. Most of existing works, however, approximate the NCP by a linear complementarity problem (LCP), where the friction cone is approximated by a polyhedral cone (Anitescu & Potra, 1997). The purpose of the approximation is to guarantee a solution with any number of contacts and contact configuration. Different methods have been proposed to compute the gradients of the solution of a LCP w.r.t. input parameters.

de Avila Belbute-Peres et al. (2018) derive these gradients using implicit differentiation in a similar way as in OptNet (Amos & Kolter, 2017). They show that the derived gradients enable end-to-end learning of unknown physics parameters such as the mass of objects.

Heiden et al. (2021b); Degrave et al. (2019); Qiao et al. (2021b) also solve collision responses based on LCP, but the LCP is solved using a projected Gauss-Seidel (PGS) method. Here the constraints of LCP are not guaranteed to hold at the end of PGS iterations, so the gradients derived by implicit differentiation might not be valid. Heiden et al. (2021b); Degrave et al. (2019) leverage existing automatic differentiation frameworks to get gradients through the PGS solver. However, significant overhead is introduced in tracing the computation graph. To improve efficiency, Qiao et al. (2021b) propose a reverse version of the PGS solver using the adjoint method.

Different from these approaches, Nimble (Werling et al., 2021) efficiently computes analytical gradients through the LCP by exploiting the sparsity of the LCP solution. Nimble also shows that analytically correct gradients might prevent an optimizer from finding a good solution and proposes an exploratory heuristic called "complementarity-aware gradient" to help optimization escape saddle points.

DiffPD (Du et al., 2021b) supports a limited LCP model which can handle only static friction. The gradients are derived analytically and sparsity is leveraged for efficiency. DiffCloth (Li et al., 2021) extends DiffPD by deriving analytical gradients of LCP with an implicit integration scheme.

2.2. Convex Optimization Models

Mujoco simulator formulates the problem of solving frictional contact impulses as a convex optimization problem (Todorov, 2011; Todorov et al., 2012; Todorov, 2014). The idea is based on maximum dissipation principle - the kinetic energy would be maximally dissipated after an inelastic collision. Then the contact impulses are solved by minimizing the post-collision kinetic energy. This is a different family of models since the complementarity condition can be violated in this formulation, i.e., the force and velocity along the contact normal direction can be simultaneously positive. In other words, LCP treats contact surfaces as hard surfaces, while convex optimization formulation treats them as soft surfaces.

With the recent progress of differentiable optimizations such as CvxpyLayer (Agrawal et al., 2019), we can easily compute the gradients of the solution of the convex optimization problem w.r.t input parameters. Zhong et al. (2021) implement this idea and demonstrate its application in end-to-end simultaneous learning of system properties, e.g., mass and potential energy, and contact properties, e.g., coefficient of friction and restitution.

2.3. Compliant Models

Compliant models assume contact surfaces can deform, which will produce elastic forces to push collided objects away from each other. Since the contact constraints are not strictly satisfied due to the soft surface assumption, compliant models are also referred to as penalty-based models from an optimization perspective. Compliant models use spring-damper systems to resolve interpenetration between surfaces. The interpenetration is usually resolved in multiple consecutive time steps and the number of time steps depends on the stiffness of the spring. In addition to normal forces, lateral friction forces are computed using either a nonlinear or a relaxed friction model for frictional contacts. In order to have a stable simulation of contact and collisions, one needs to carefully tune the parameters such as spring stiffness, and these parameters could be hard to tune in a contact-rich scenario.

Since the forces from the spring-damper system are continuously differentiable functions of position and velocity, the trajectories of position and velocity are also continuous and differentiable. This makes compliant models easy to implement using existing automatic differentiation tools. A number of works have explored differentiable simulation with compliant models, including (Giftthaler et al., 2017), (Carpentier & Mansard, 2018), (Xu et al., 2022), NeuralSim (Heiden et al., 2021b), gradSim (Murthy et al., 2021), ADD (Geilinger et al., 2020), IPC (Li et al., 2020), DiSECt (Heiden et al., 2021a), DiffPD (Du et al., 2021b), Warp (Macklin, 2022) and the legacy implementation of Brax (Geilinger et al., 2020).

2.4. Position-based Dynamics

To resolve contacts, compliant models manipulate *forces*, LCP as well as convex optimization models manipulate *velocities*, and position-based dynamics (PBD) (Müller et al., 2007) directly manipulate *positions*. PBD is originally proposed to tackle contact-rich physics-based animation in computer graphics and games. In PBD, interpenetration in a contact event is resolved by directly projecting points to valid locations in such a way that takes into account the conservation of linear and angular momentum. Velocities are then updated based on the updated positions and the positions in the previous time step. Extended PBD (XPBD) (Macklin et al., 2016) extends the original PBD to address the problems of iteration-dependent contact stiffness by introducing elastic potentials.

The forward pass of calculating position-based impulses only involves differentiable operations so the gradients can be computed by automatic differentiation. Open source libraries such as Warp (Macklin, 2022) and Brax (Freeman et al., 2021) implement differentiable PDB in this way.

Liang et al. (2019) study differentiable cloth simulation and formulate the updates of positions as a quadratic programming (QP) problem. They introduce a QR decomposition step after the implicit differentiation to compute the gradient through the QP more efficiently in the context of cloth simulation. Qiao et al. (2020) adopt a similar approach but use generalized coordinates instead of Cartesian coordinates and additionally deal with the mapping between the two coordinates when deriving gradients. A relevant work in learning contact constraints (Yang et al., 2020) also use position-based techniques to handle inelastic contacts.

2.5. Other Related Works

We further review some other relevant differentiable physics simulators. Macklin et al. (2020) propose a primal/dual descent method for simulation, where the primal formulation is related to projective dynamics (Bouaziz et al., 2014) and the dual formulation is related to XPBD. They demonstrate the differentiability of the primal formulation using an example of trajectory optimization. Le Lidec et al. (2021) formulate the frictional contact problem into a sequence of QCQPs. The analytical gradients are derived by implicit differentiation. They demonstrate the framework on system identification from videos of dynamical scenes. Chen et al. (2021) propose neural event functions to model instantaneous velocity change during a collision. They show that for frictionless contacts, both the neural event functions and the instantaneous updates can be learned. Sutanto et al. (2020) demonstrate the learning of physics parameters by encoding physical constraints in differentiable simulation. However, simulation with contacts has not been investigated.

3. Implementation Choices for Experiments

In principle, different concepts of differentiable simulation mentioned above are not restricted to any single software tool. For example, we can implement them in general-purpose machine learning tools such as Tensorflow, Pytorch and Jax, or tools that are tailored for physics simulations such as DiffTaichi (Hu et al., 2020) and Warp (Macklin, 2022). In this work, we implement different differentiable contact formulations on three systems by leveraging existing open source tools to avoid reinventing the wheel. Our implementation choice for each formulation is detailed below.

For **LCPs**, we implement our systems in Nimble (Werling et al., 2021). Nimble is a fork of the DART physics engine (Lee et al., 2018), with analytical gradients of LCP and PyTorch binding. DiffTaichi (Hu et al., 2020) has pointed out that directly adding velocity impulse can lead to incorrect gradients and proposed using continuous-time detection or time-of-impact (TOI) for computing correct gradients. Since LCP uses velocity impulse in simulation, it would suffer from this problem if the correction is not considered. Nimble has implemented continuous-time detection and therefore does not suffer from this particular problem, as we can see in the experiment sections. In this work we focus on frictionless contact, where the NCP and LCP formulations become equivalent. We leave the investigation of NCP formulations (Howell et al., 2022) as a future work.

For **convex optimization models,** we implement our systems with a simplified version of DiffCoSim (Zhong et al., 2021). As convex optimization models also calculate velocity impulses and the original DiffCoSim does not implement TOI, we add the TOI implementation for comparison.

For **compliant models** and **PBD**, we implement our systems with both Brax (Freeman et al., 2021) and Warp (Macklin, 2022). The authors of Brax have experimented with compliant models but encountered stability issues. Nevertheless, they include this implementation in the project (the `legacy_spring` dynamics mode). Their latest collision model is position-based (the `pbd` dynamics mode), which also supports frictional and elastic contacts. Warp is a recently released open source project for high-performance physics simulation. They provide example implementations of compliant models and PBD along with the release, but their current PBD implementation is preliminary and does not support frictional and elastic contact yet. We add frictional and elastic support tailored to our systems.

For frictionless collisions, we can easily compute the velocity impulse without using the LCP or convex optimization models. In fact, the `billiards` example in diffTaichi (Hu et al., 2020) is implemented in this way. In our systems with frictionless collisions, we also implement direct computation of velocity impulses using diffTaichi.

Table 1. Task 1: gradients of the final heights w.r.t. initial position, velocity and control

Implementations		$\partial p_{y,N}/\partial p_{y,0}$	$\partial p_{y,N}/\partial v_{y,0}$	$\partial p_{y,N}/\partial u_{y,0}$
Analytical gradients		$\mathbf{-1.0000}$	$\mathbf{-1.0000}$	$\mathbf{-0.0021}$
LCP (with TOI)		$\mathbf{-1.0000}$	$\mathbf{-1.0000}$	$\mathbf{-0.0021}$
Convex Optimization Model	with TOI	$\mathbf{-1.0000}$	$\mathbf{-1.0000}$	$\mathbf{-0.0021}$
	without TOI	1.0000	-0.0958	-0.0002
Direct Velocity Impulse	with TOI	$\mathbf{-1.0000}$	$\mathbf{-1.0000}$	$\mathbf{-0.0021}$
	without TOI	1.0000	-0.1000	-0.0002
Compliant Model	Warp	-1.5680	-1.2248	-0.0026
	Brax	1.0000	-0.0958	-0.0004
PBD	Warp	0.0000	-0.5479	-0.0011
	Brax	-0.0020	-0.5467	-0.0023

4. Experiments

In this section, we investigate and compare the performance of differentiable contact models on three tasks. The physics system in all three tasks are two-dimensional. We mainly work with a discrete-time formulation, where the simulation duration T is discretized into N time steps with $\Delta t = T/N$. In all three tasks, we choose $\Delta t = 1/480s$. We use $p_n = [p_{x,n}, p_{y,n}]$ to denote the position of an object at the nth time step. Task 3 involves two objects and the configuration of the system at the nth time step is denoted as $p_n = [p_{1,n}, p_{2,n}] = [p_{x_1,n}, p_{y_1,n}, p_{x_2,n}, p_{y_2,n}]$. When working in a continuous-time perspective, we use $p(t)$ to denote the system configuration at time t. The velocity variables are denoted by v and defined similarly.

Gradients with control. From a continuous-time viewpoint, the concept of gradients w.r.t. the position or velocity at certain time (this paper mainly examines the quantity at the initial time) is well-defined while the gradients w.r.t. the control is more subtle. The reason is that when the control is viewed as a function of time, we need to extend the classical concept of gradients to some functional derivatives, such as Gateaux derivative or Fréchet derivative (Gelfand et al., 2000). To avoid the complication of diving into deeper mathematical concepts, here we examine the gradients w.r.t. the control in a special setting. Given $\Delta t > 0$ and continuous-time control profile $\tilde{u}(t)$ defined on $[0, T]$, we apply constant u_0 on $[0, \Delta t]$ and then $\tilde{u}$ on $[\Delta t, T]$ to get the total loss l. Then $\partial l/\partial u_0$ can be defined in the classical sense and we treat that as the analytical gradient for comparison. We remark that, in the finite-dimensional case, the functional derivative is consistent with the classical derivative up to a time discretization factor. Specifically, when $u_0 = \tilde{u}(0)$, the gradient we consider converges to the functional derivative at $t = 0$ as $\Delta t \to 0$.[1]

4.1. Task 1: Gradients with a Simple Collision

In this section, we revisit the task of simple collision studied in DiffTaichi. As shown in Figure 1, this is a 2D system without gravity. The initial position and velocity of the ball are $p_0 = [-1, 1]$ and $v_0 = [2, -2]$. We add constant zero controls $u_n = [0, 0], n = 0, ..., N-1$ to the ball in order to compute the gradients w.r.t. controls. The simulation duration is $T = 1s$. During the simulation, the ball has a perfectly elastic frictionless collision with the ground. We are interested in the gradients of the final height w.r.t. the initial position, velocity and control.

In this example, we can write down the analytical expression

[1] We take the functional $l(u) = \int_0^T u(t)^2 \mathrm{d}t$ as an example. The Fréchet derivative of $l(u)$ in L^2 is $\frac{\delta l}{\delta u}(t) = 2u(t)$. If we take the discrete approximation of the integral as $\tilde{l}(u_0, \cdots, u_{N-1}) = \sum_{i=0}^{N-1} u_i^2 \Delta t$, where u_i are values at $i\Delta t$, the function $\tilde{l}$ has gradient $\nabla \tilde{l}(u_0, \cdots, u_{N-1}) = 2\Delta t(u_0, \cdots, u_{N-1})$.

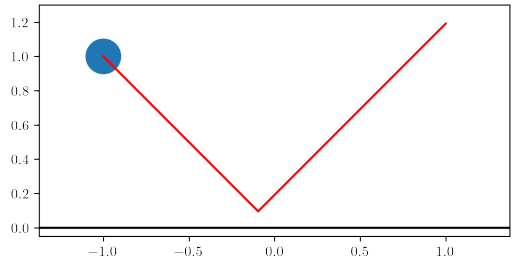

Figure 1. Simple collision in task 1.

Table 2. Task 2: gradients of loss w.r.t. initial velocity; optimized velocity; and trajectory mode.

Implementations		$\partial l/\partial v_{x,0}, \partial l/\partial v_{y,0}$ at iteration 0	$v_{x,0}, v_{y,0}$ at iteration 999	trajectory mode
LCP (with TOI)		$-3.2016, -0.3059$	$10.2297, -4.1396$	Trajectory 1
Convex Optimization Model	with TOI	$-3.1652, -0.3059$	$10.2575, -4.2041$	Trajectory 1
	without TOI	$1.5898, -0.1391$	$-2.4934, -4.3371$	Trajectory 2
Direct Velocity Impulse	with TOI	$-3.1662, -0.3111$	$10.1841, -4.1606$	Trajectory 1
	without TOI	$1.5550, -0.1552$	$-2.4933, -4.2984$	Trajectory 2
Compliant Model	Warp	$-4.9727, -0.3984$	$10.6386, -4.2821$	Trajectory 1
	Brax	$1.6971, -0.0366$	$-2.4935, -4.7927$	Trajectory 2
PBD	Warp	$-16.7149, -2.0866$	$9.7299, -4.0706$	Trajectory 1
	Brax	$-0.8311, -0.0663$	$10.4865, -4.7518$	Trajectory 1

of the final height (see Appendix A for a derivation),

$$p_{y,N} = -p_{y,0} - v_{y,0}T - u_{y,0}(T\Delta t - \frac{1}{2}(\Delta t)^2) + 2r, \quad (1)$$

where r is the radius of the ball. In Table 1, we compare the analytical gradients with gradients computed by different implementations. We find that the three implementations that can compute accurate gradients in this example are the ones that implement TOI. This is consistent with DiffTaichi's observation. It should be emphasized that discarding TOI in velocity-impulse-based contact models will result in a completely wrong gradient w.r.t. the position, and making Δt smaller cannot solve the issue. We refer the interested reader to DiffTaichi (Hu et al., 2020) for more details.

For compliant models, there exists no concept of TOI since interpenetration is resolved in multiple time steps. The gradients w.r.t. position of the Warp and Brax implementations do not match. In fact, they are even in the opposite directions. This phenomenon might be due to different implementation details, e.g. spring stiffness.

The two PBD implementations agree well, but they do not match the analytical gradients. In particular, the gradients w.r.t. position are close to zero. This is because when a collision (interpenetration) is detected, the position of the ball is updated to resolve the interpenetration. An infinitesimal change in $p_{y,0}$ will not change the value of $p_{y,n}$ after the collision, since the ball is always updated to touch the ground ($p_y = r$) right after the collision.

We also remark that across all implementations, the gradients w.r.t. velocity and control are all negative. In other words, even if they might be wrong in value, they are correct in direction. If we are optimizing over velocity or control in an optimization task, it is possible that it ends up with a reasonable solution with these "inaccurate" gradients.

4.2. Task 2: Optimize the Initial Velocity of a Bouncing Ball to Hit a Target

In this section, we revisit a task studied in Warp (Macklin, 2022) and (Macklin et al., 2020). As shown in Figure 2(a), we have a ball of radius $r = 0.1$ with initial position $p_0 = [-0.5, 1.0]$ and initial velocity $v_0 = [5, -5]$. The simulation duration is $T = 0.6s$.

The initial trajectory is shown in Figure 2(a). We assume collisions are frictionless and the elastic coefficient is $e = 0.92$. These contact properties will produce trajectories close to the original Warp implementation.[2]

The task is to optimize the initial velocity such that the ball hits a target at $p_{\text{target}} = [-2.0, 1.5]$ at the end of the simulation. With differentiable simulations, we can set up a loss function $l = ||p_N - p_{\text{target}}||_2^2$, get the gradient w.r.t the initial velocity $\partial l/\partial v_{x,0}, \partial l/\partial v_{y,0}$ and use gradient descent to update the initial velocity to minimize the loss. In this task, we use a learning rate of 0.01 for 1000 gradient steps.

The learning curves in Figure 2(d) (left) indicate that all the implementations can successfully minimize the loss to zero and accomplish the task. Table 2 shows the gradients of loss w.r.t. initial velocity before optimization (Figure 2(a)) and the optimized initial velocity after 1000 gradient steps. We observe that this task has at least two solutions, with the trajectories shown in Figure 2(b) and 2(c). Different implementations learn different trajectories, as summarized in the last column in Table 2.

[2]The original Warp implementation is frictionless and use the compliant model with nonzero damping coefficient. In this non-perfectly elastic collision case, there's no one-to-one mapping between compliant model parameters and the elastic coefficient in other models. We examine the horizontal velocity of the ball before (5.0) and after (-4.6) the bouncing with the wall in the original trajectory and choose to use an elastic coefficient $e = |-4.6/5.0| = 0.92$ to produce similar trajectories.

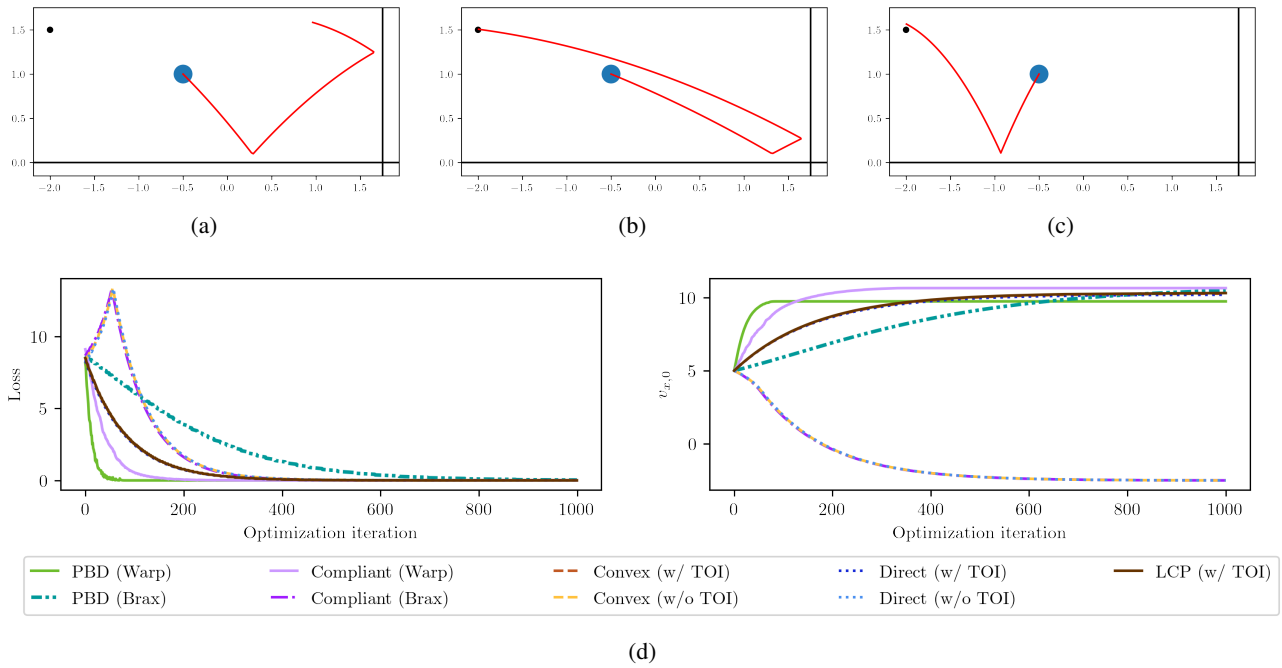

Figure 2. Trajectories and learning curves of task 2. **(a)** initial trajectory; **(b)** optimized trajectory 1; **(c)** optimized trajectory 2; **(d) Left**: learning curves - loss over iterations; **Right**: learning curves - initial horizontal velocity over iterations.

We also find that which solution an implementation end up with can be inferred from the sign of $\partial l / \partial v_{x,0}$ at iteration 0. For example, the LCP implementation results in $\partial l / \partial v_{x,0} < 0$. To minimize the loss, the algorithm increases the value of $v_{x,0}$ and ends up with trajectory 1. For those implementations with $\partial l / \partial v_{x,0} > 0$, the algorithm decreases the value of $v_{x,0}$ and ends up with trajectory 2. This reasoning can be verified in Figure 2(d) (right).

If we compare implementations with and without TOI, we notice that TOI affects the sign of $\partial l / \partial v_{x,0}$, which in turn affects the optimized trajectory. The two compliant model implementations again produce gradients in the opposite directions.

The takeaway from this task is that even in a simple setting with two frictionless collisions, the gradients computed by different implementations do not agree. These differences have a huge impact on optimization and can lead to totally different outcomes. We also experiment with frictional contacts in this task and have similar observations (see Appendix B.)

4.3. Task 3: Learning Optimal Control with a Two-ball Collision

In this section, we investigate the learning of optimal control sequences using differentiable simulation. The specific task has been studied by Hu et al. (2022) using a different approach, i.e., hybrid minimum principle (HMP). As Hu et al. (2022) has derived the analytical solution of this specific task, we are able to use that to measure the performance of different implementations.

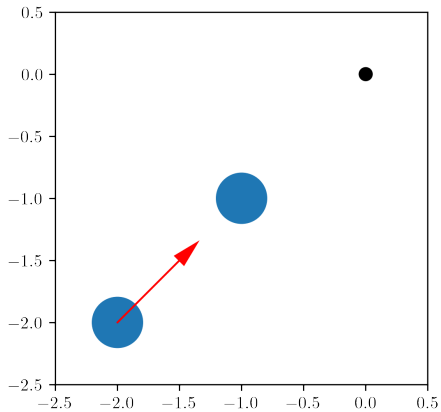

Figure 3. The initial configuration and target position of task 3.

The system is shown in Figure 3. We have two balls, of the same size (radius $r = 0.2$) on a plane with no gravity. The initial positions of the balls are $p_{1,0} = [-2, -2]$ and $p_{2,0} = [-1, -1]$ and the initial velocities are $v_{1,0} = v_{2,0} = [0, 0]$. The simulation duration is $T = 1s$. We are able to add control inputs as forces acted on the first ball. The goal

Table 3. Task 3: gradients of loss w.r.t. initial position, velocity and control at iteration 0

Implementations		$\partial l/\partial p_{x_1,0}, \partial l/\partial p_{x_2,0}$ at iteration 0	$\partial l/\partial v_{x_1,0}, \partial l/\partial v_{x_2,0}$ at iteration 0	$\partial l/\partial u_{x_1,0}$ at iteration 0
Analytical gradients		$\mathbf{-0.3987, -0.3213}$	$\mathbf{-0.4978, -0.2221}$	$\mathbf{-0.0009}$
LCP (with TOI)		$-0.5476, -0.1825$	$-0.6031, -0.1270$	0.0000
Convex Optimization Model	with TOI	$-0.7325, 0.0000$	$-0.7310, -0.0015$	-0.0003
	without TOI	$0.0000, -0.7291$	$-0.2233, -0.5058$	0.0008
Direct Velocity Impulse	with TOI	$-0.7191, 0.0000$	$-0.7191, 0.0000$	-0.0002
	without TOI	$0.0000, -0.7156$	$-0.2221, -0.4935$	0.0008
Compliant Model	Warp	$0.9509, -1.6969$	$0.4348, -1.1808$	0.0022
	Brax	$0.0000, -0.7252$	$-0.2220, -0.5031$	0.0016
PBD	Warp	$-0.3610, -0.3610$	$-0.4723, -0.2497$	0.0003
	Brax	$-0.3613, -0.3606$	$-0.4725, -0.2494$	0.0005

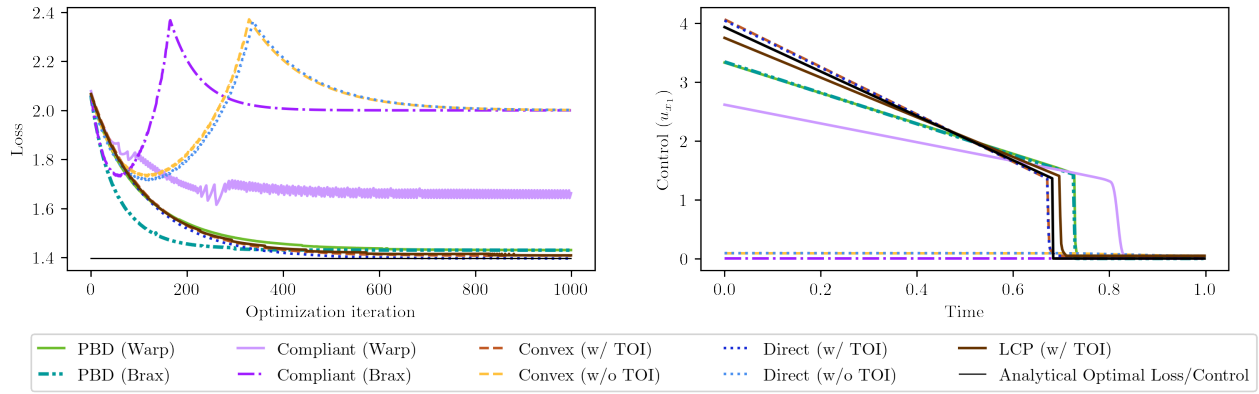

Figure 4. Results of task 3. **Left**: learning curves; **Right**: learned control profiles along with analytical optimal control profile.

in this task is to push ball 1 to strike ball 2 so that ball 2 will be close to the origin at the end of the simulation.

The problem can be formulated as an optimal control problem in continuous-time with state jumps:

$$\underset{u(\cdot)}{\text{minimize}} \quad \phi(s(T)) + \int_0^T L(s(t), u(t))\mathrm{d}t, \tag{2}$$

$$\text{subject to} \quad \dot{s}(t) = f(s(t), u(t)), t \in [0, \gamma) \text{ or } t \in (\gamma, T], \tag{3}$$

$$\psi(s(\gamma^-)) = 0, \tag{4}$$

$$s(\gamma^+) = g(s(\gamma^-)). \tag{5}$$

Here we use $s = [p, v]$ to denote the positions and velocities of two balls as the state variable. f denotes the state dynamics under external forces u; γ denotes the time of collision between two balls, characterized by the distance function ψ between two balls; and g denotes the effect of collision on the state. We choose terminal cost to be $\phi(s(T)) = ||p_2(T)||_2^2$ to capture our goal and running cost to be $L(s, u) = \epsilon||u||_2^2$ with $\epsilon = 0.1$ to penalize large control inputs. See the appendix of Hu et al. (2022) for the optimal solution of this problem in an analytic form.

To solve the above problem approximately, we discrete the problem into

$$\underset{u_0,\ldots,u_{N-1}}{\text{minimize}} \quad \phi(s_N) + \sum_{i=0}^{N-1} L(s_i, u_i)\Delta t, \tag{6}$$

$$\text{subject to} \quad s_{i+1} = \texttt{step}(s_i, u_i, \Delta t). \tag{7}$$

The `step` function takes the current state and control as inputs and calculates the next time step state based on dynamics and collisions. With differentiable simulations, we can differentiate through the `step` function and solve for the optimal control sequence directly using gradient descent.

We initiate our control sequence as a constant force $u_n = [3, 3], n = 0, ..., N-1$. With this constant control sequence,

we can compute the gradients of the loss w.r.t. initial positions and velocities of the two balls as well as the control at the first time step. As this task is symmetric in x and y coordinates, we only present the x-components in Table 3. The y-components are the same as the corresponding x-components. The analytical gradients in the table are computed by deriving the analytical expression of the loss as done in Section 4.1. We provide code scripts of computing these gradients in Appendix C.

Surprisingly, from Table 3, we observe that none of the gradients from differentiable simulators match the analytical gradients; only the two implementations based on PBD give results that are close to the analytical gradients.

In this task, we use a learning rate of 10 for 1000 gradient steps. Figure 4 (left) shows the learning curves of different implementations along with the analytical optimal loss (1.3965). We observe that two implementations without TOI and two compliant model implementations fail to converge to the analytical optimal loss. The rest of the implementations converge to values that are close to the analytical optimal loss.

Figure 4 (right) compares the control profile optimized by different simulators with the analytical optimal control, based on the analytical expression presented in (Hu et al., 2022). We observe that for two implementations without TOI and the Brax implementation of the compliant model, the learned control sequences are close to zero all the time. Under a zero control sequence, the two balls would not move at all resulting in a running loss of 0 and a terminal loss of 2. From Figure 4 (left) we can confirm that these three implementations end up with a loss close to 2. For all the other implementations, the shapes of the learned optimal control profiles resemble the analytical one, where, they linearly decrease before the collision and drop to zero after the collision. Three implementations with TOI perform the best in learning the optimal control sequence. We remark that, as reported in (Hu et al., 2022), the deep reinforcement learning algorithm PPO (Schulman et al., 2017) usually finds a solution with a running loss of 0 and a terminal loss of 2, if there is no reward shaping. Compared to such model-free reinforcement learning methods, optimizing with differentiable physics simulations demonstrates its strength in solving control tasks.

We also experiment with the spring stiffness parameter in the compliant models, as shown in Figure 5. We observe that a small stiffness (100) makes the surface too soft to represent the actual collision phenomenon. The initial loss is 2.39 in this case while the analytical loss before optimization is around 2.06. The initial losses of larger stiffness are indeed around 2.06, but a large stiffness such as 10^5 makes learning unstable and fails to learn a reasonable control strategy.

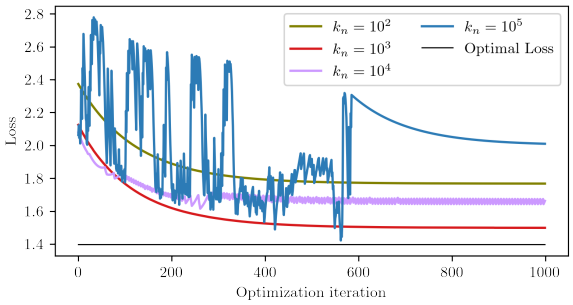

Figure 5. Task 3: learning curves of compliant models implemented in Warp with different spring stiffness k_n. Spring stiffness determines the normal contact force based on interpenetration d, i.e., $f_n = -k_n \cdot d$.

The takeaway from this task is that the gradients computed by differentiable physics simulators might not reflect the true gradients in the physics process. Nevertheless, they might still be helpful in gradient-based learning tasks. The reasons behind the successful optimizations with wrong gradients need further investigation. In more complex contact-rich scenarios, it is likely that none of the current differentiable simulators can accomplish particular optimization tasks.

5. Conclusion

In this paper, we investigated gradient computation using existing differentiable physics simulation tools. We apply multiple differentiable simulators on three tasks. All the three tasks only involve simple frictionless collision and do not involve ill-cases such as grazing contacts (Corner, 2017), which are likely not differentiable. We find that in a specific system (Task 3), the gradients w.r.t. position, velocity and control computed from all differentiable simulators studied in this paper do not match the analytical result well.

This finding raises two questions: 1) how can the optimization task be successfully achieved with wrong gradients? and more importantly 2) how to improve differentiable simulations to compute correct gradients? If these questions are properly addressed, differentiable physics simulation can serve as a powerful interpretable tool in end-to-end optimization tasks, such as system identification, learning of dynamics systems, learning of optimal control, geometry optimization and reinforcement learning. We hope this work can motivate future research into differentiable physics simulations. A recent study (Suh et al., 2022) compares the gradients of a differentiable simulator and policy gradients in a stochastic setting. It will also be of interest to extend the comparison with the differentiable simulator with improved gradients.

References

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Anitescu, M. and Potra, F. A. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997.

Bouaziz, S., Martin, S., Liu, T., Kavan, L., and Pauly, M. Projective dynamics: Fusing constraint projections for fast simulation. *ACM transactions on graphics (TOG)*, 33(4):1–11, 2014.

Carpentier, J. and Mansard, N. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and systems (RSS 2018)*, 2018.

Chen, R. T. Q., Amos, B., and Nickel, M. Learning neural event functions for ordinary differential equations. In *International Conference on Learning Representations*, 2021.

Corner, S. M. *Modeling, sensitivity analysis, and optimization of hybrid, constrained mechanical systems*. PhD thesis, Virginia Polytechnic Institute and State University, 2017.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., and Kolter, J. Z. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31, 2018.

Degrave, J., Hermans, M., Dambre, J., et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, pp. 6, 2019.

Du, T., Hughes, J., Wah, S., Matusik, W., and Rus, D. Underwater soft robot modeling and control with differentiable simulation. *IEEE Robotics and Automation Letters*, 6(3): 4994–5001, 2021a.

Du, T., Wu, K., Ma, P., Wah, S., Spielberg, A., Rus, D., and Matusik, W. Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics (TOG)*, 41(2):1–21, 2021b.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax.

Geilinger, M., Hahn, D., Zehnder, J., Bächer, M., Thomaszewski, B., and Coros, S. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

Gelfand, I. M., Silverman, R. A., et al. *Calculus of variations*. Courier Corporation, 2000.

Giftthaler, M., Neunert, M., Stäuble, M., Frigerio, M., Semini, C., and Buchli, J. Automatic differentiation of rigid body dynamics for optimal control and estimation. *Advanced Robotics*, 31(22):1225–1237, 2017.

Heiden, E., Macklin, M., Narang, Y., Fox, D., Garg, A., and Ramos, F. Disect: A differentiable simulation engine for autonomous robotic cutting. *arXiv preprint arXiv:2105.12244*, 2021a.

Heiden, E., Millard, D., Coumans, E., Sheng, Y., and Sukhatme, G. S. Neuralsim: Augmenting differentiable simulators with neural networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9474–9481. IEEE, 2021b.

Howell, T. A., Cleac'h, S. L., Kolter, J. Z., Schwager, M., and Manchester, Z. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 2022.

Hu, W., Long, J., Zang, Y., E, W., and Han, J. Solving optimal control of rigid-body dynamics with collisions using the hybrid minimum principle. *arXiv preprint arXiv:2205.08622*, 2022.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., and Durand, F. Difftaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2020.

Le Lidec, Q., Kalevatykh, I., Laptev, I., Schmid, C., and Carpentier, J. Differentiable simulation for physical system identification. *IEEE Robotics and Automation Letters*, 6 (2):3413–3420, 2021.

Lee, J., Grey, M. X., Ha, S., Kunz, T., Jain, S., Ye, Y., Srinivasa, S. S., Stilman, M., and Liu, C. K. Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500, 2018.

Li, M., Ferguson, Z., Schneider, T., Langlois, T., Zorin, D., Panozzo, D., Jiang, C., and Kaufman, D. M. Incremental potential contact: Intersection-and inversion-free, large-deformation dynamics. *ACM transactions on graphics*, 2020.

Li, Y., Du, T., Wu, K., Xu, J., and Matusik, W. Diffcloth: Differentiable cloth simulation with dry frictional contact. *arXiv preprint arXiv:2106.05306*, 2021.

Liang, J. and Lin, M. C. Differentiable Physics Simulation. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Liang, J., Lin, M., and Koltun, V. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 32, 2019.

Macklin, M. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

Macklin, M., Müller, M., and Chentanez, N. Xpbd: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, pp. 49–54, 2016.

Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S., and Kim, T. Primal/dual descent methods for dynamics. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–12, 2020.

Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

Murthy, J. K., Macklin, M., Golemo, F., Voleti, V., Petrini, L., Weiss, M., Considine, B., Parent-Lévesque, J., Xie, K., Erleben, K., Paull, L., Shkurti, F., Nowrouzezahrai, D., and Fidler, S. gradsim: Differentiable simulation for system identification and visuomotor control. In *International Conference on Learning Representations*, 2021.

Qiao, Y., Liang, J., Koltun, V., and Lin, M. Differentiable simulation of soft multi-body systems. *Advances in Neural Information Processing Systems*, 34, 2021a.

Qiao, Y.-L., Liang, J., Koltun, V., and Lin, M. Scalable differentiable physics for learning and control. In *International Conference on Machine Learning*, pp. 7847–7856. PMLR, 2020.

Qiao, Y.-L., Liang, J., Koltun, V., and Lin, M. C. Efficient differentiable simulation of articulated bodies. In *International Conference on Machine Learning*, pp. 8661–8671. PMLR, 2021b.

Rojas, J., Sifakis, E., and Kavan, L. Differentiable implicit soft-body physics. *arXiv preprint arXiv:2102.05791*, 2021.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Song, C. and Boularias, A. Identifying mechanical models of unknown objects with differentiable physics simulations. In *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pp. 749–760. PMLR, 2020a.

Song, C. and Boularias, A. Learning to slide unknown objects with differentiable physics simulations. In *Robotics science and systems*, 2020b.

Strecke, M. and Stueckler, J. Diffsdfsim: Differentiable rigid-body dynamics with implicit shapes. In *2021 International Conference on 3D Vision (3DV)*, pp. 96–105. IEEE, 2021.

Suh, H., Simchowitz, M., Zhang, K., and Tedrake, R. Do differentiable simulators give better policy gradients? *arXiv preprint arXiv:2202.00817*, 2022.

Sutanto, G., Wang, A., Lin, Y., Mukadam, M., Sukhatme, G., Rai, A., and Meier, F. Encoding physical constraints in differentiable newton-euler algorithm. In *Learning for Dynamics and Control*, pp. 804–813. PMLR, 2020.

Todorov, E. A convex, smooth and invertible contact model for trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation*, pp. 1071–1076, 2011.

Todorov, E. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6054–6061, 2014.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Werling, K., Omens, D., Lee, J., Exarchos, I., and Liu, C. K. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021.

Xu, J., Chen, T., Zlokapa, L., Foshey, M., Matusik, W., Sueda, S., and Agrawal, P. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.

Xu, J., Macklin, M., Makoviychuk, V., Narang, Y., Garg, A., Ramos, F., and Matusik, W. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZSKRQMvttc.

Yang, S., He, X., and Zhu, B. Learning physical constraints with neural projections. *Advances in Neural Information Processing Systems*, 33:5178–5189, 2020.

Zhong, Y. D., Dey, B., and Chakraborty, A. Extending lagrangian and hamiltonian neural networks with differentiable contact models. *Advances in Neural Information Processing Systems*, 34, 2021.

Appendices

A. Derivation of Equation 1

For this system, the dynamics along x and y axes are decoupled. Here we are interested in the dynamics along y axis. We assume the initial control component $u_{y,0}$ is applied to the ball from time 0 to time Δt, and the control remain zero afterwards. Then the velocity and position components at time Δt are

$$v_{y,1} = v_{y,0} + u_{y,0}\Delta t \tag{8}$$

$$p_{y,1} = p_{y,0} + v_{y,0}\Delta t + \frac{1}{2}u_{y,0}(\Delta t)^2 \tag{9}$$

Since the contact is perfectly elastic, we have

$$(p_{y,N} - r) + (p_{y,1} - r) = -v_{y,1}(T - \Delta t) \tag{10}$$

Thus we get Equation 1

$$p_{y,N} = -p_{y,0} - v_{y,0}T - u_{y,0}(T\Delta t - \frac{1}{2}(\Delta t)^2) + 2r. \tag{11}$$

The analytical gradients are

$$\frac{\partial p_{y,N}}{\partial p_{y,0}} = -1, \qquad \frac{\partial p_{y,N}}{\partial v_{y,0}} = -T, \qquad \frac{\partial p_{y,N}}{\partial u_{y,0}} = -T\Delta t + \frac{1}{2}(\Delta t)^2 \tag{12}$$

B. Task 2 with Friction

We change the frictionless contacts in Task 2 to be frictional with coefficient $\mu = 0.1$. In this case, it would be challenging to directly compute the velocity impulse. Thus, we implement the all the other contact model formulations and present the result in Table 4. Here Trajectory 1 and 2 refers to trajectories similar to those in Figure 2(b) (colliding with both the ground and the wall) and Figure 2(c) (colliding with only the ground), respectively.

Table 4. Task 2 with friction: gradients of loss w.r.t. initial velocity; optimized velocity; and trajectory mode.

Implementations		$\partial l/\partial v_{x,0}, \partial l/\partial v_{y,0}$ at iteration 0	$v_{x,0}, v_{y,0}$ at iteration 999	trajectory mode
LCP (with TOI)		$-3.2016, -0.3190$	$11.3389, -6.1378$	Trajectory 1
Convex Optimization Model	with TOI	$-4.2416, -1.4351$	$10.3454, -3.9920$	Trajectory 1
	without TOI	$3.2069, 0.1918$	$0.6268, -4.7243$	N/A (Failed)
Direct Velocity Impulse		N/A		
Compliant Model	Warp	$-4.5909, -1.4180$	$11.8168, -6.4319$	Trajectory 1
	Brax	$1.7889, -0.0147$	$-2.5777, -4.7980$	Trajectory 2
PBD	Warp	$-12.9160, 126.9161$	$10.3663, -3.9202$	Trajectory 1
	Brax	$-0.8622, -0.3825$	$10.1879, -3.9497$	Trajectory 1

C. Code Snippets for Computing Analytical Gradients in Task 3

The gradients in this case can be derived in an analytical form. They can also be computed by automatic differentiation. As the analytical expressions are lengthy, here we provide two code snippets to compute the these gradients in JAX. We have verified that the gradients computed by analytical expressions and by the code snippets match with each other as expected.

Since this problem is symmetric, the gradients in the direction $[1, -1]$ are zero. In the code snippets, we simplify the problem into a 1D problem by computing the gradients along the direction $[1, 1]$. We then convert them back to the 2D coordinate frame.

The first code snippet only computes the gradient w.r.t. initial position, it has a simple form and should be easy to understand

```
import jax
import jax.numpy as jnp
import numpy as np
def loss_fn(x, u_c=3*jnp.sqrt(2), r=0.2, T=1.):
    x1_0 = x[0]
    x2_0 = x[1]
    # time of collision
    s = jnp.sqrt(2 * (x2_0 - x1_0 - 2 * r) / u_c)
    # velocity at time of collision
    v1_s = u_c * s
    x2_T = x2_0 + v1_s * (T - s)
    l = x2_T ** 2
    return l, (s, v1_s, x2_T)

grad_loss_fn = jax.grad(loss_fn, has_aux=True)
x0 = jnp.array([-2 * jnp.sqrt(2), -1 * jnp.sqrt(2)])
dl_dx, aux_data = grad_loss_fn(x0)
print((dl_dx)/ jnp.sqrt(2))

# output
# [-0.39866853 -0.3212531 ]
```

Listing 1. computing gradients w.r.t. initial position in Task 3

The second code snippet computes the gradient w.r.t. initial position, velocity and control.

```
import jax
import jax.numpy as jnp
import numpy as np
def loss_fn(x0, v0, u0, u_c=3*jnp.sqrt(2), dt=1./480, r=0.2, T=1., epsilon=0.1):
    x1_0 = x0[0] ; x2_0 = x0[1]
    v1_0 = v0[0] ; v2_0 = v0[1]
    # integrate first time analytically
    v1_dt = v1_0 + u0 * dt ; v2_dt = v2_0
    x1_dt = x1_0 + v1_0 * dt + u0 * dt**2/2
    x2_dt = x2_0 + v2_0 * dt
    # solve time of collision
    # \int_{dt}^{s} (v1_dt + u_c*(t-dt) - v2_dt) = x2_dt - x1_dt - 2 * r
    dist_dt = x2_dt - x1_dt - 2 * r
    # a (s-dt)^2 + b (s-dt) + c = 0
    a = u_c / 2
    b = v1_dt - v2_dt
    c = -dist_dt
    s = (-b + jnp.sqrt(b*b - 4*a*c)) / (2*a) + dt
    # velocity at time of collision
    v1_s = v1_dt + u_c * (s - dt)
    x2_s = x2_dt + v2_dt * (s - dt)
    x2_T = x2_s + v1_s * (T - s)
    l = x2_T ** 2 + epsilon * u0 * dt # running loss for future us does not matter
    return l, (s, v1_s, x2_T)

grad_loss_fn = jax.grad(loss_fn, [0, 1, 2], has_aux=True)
x0 = jnp.array([-2 * jnp.sqrt(2), -1 * jnp.sqrt(2)])
v0 = jnp.array([0., 0.])
u0 = 3 * jnp.sqrt(2)
dl, aux_data = grad_loss_fn(x0, v0, u0)
dl_dx0, dl_dv0, dl_du0 = dl
print(dl_dx0 / jnp.sqrt(2))
print(dl_dv0 / jnp.sqrt(2))
print(dl_du0 / jnp.sqrt(2))
# output
# [-0.39866856 -0.32125315]
# [-0.49779078 -0.22213092]
```

```
# -0.0008888851
```

Listing 2. computing gradients w.r.t. initial position velocity and control in Task 3