phiflow: A Differentiable PDE Solving Framework for Deep Learning via Physical Simulations

Philipp Holl
Technical University of Munich
philipp.holl@tum.de

Vladlen Koltun
Intel Labs
vladlen.koltun@intel.com

Kiwon Um
LTCI, Telecom Paris, IP Paris
kiwon.um@telecom-paris.fr

Nils Thuerey
Technical University of Munich
nils.thuerey@tum.de

1 Introduction

Understanding physical environments is a key requirement for machine learning applications such as autonomous agents and robots [8, 1]. It is typically of vital importance to not only understand the unperturbed physical behavior but also anticipate how the environment reacts to an agent interacting with it [15, 6]. We consider partial differential equations (PDEs) as the most fundamental description of physical systems. The language of PDEs is general enough to describe every physical theory, from quantum mechanics and general relativity to turbulent flows [14]. Existing machine learning methods that deal with agents learning to interact with their environments have often focused on reinforcement learning [11, 5], but for high-dimensional environments, the computational cost of exploring the state space puts severe limits on the number of interaction parameters with which the agent can influence the physical system [9].

Meanwhile, progress has been made in utilizing differentiable solvers to find solutions to high-dimensional optimization problems [15, 4, 13]. Yet existing methods are still computationally expensive and thus limited to short time frames. We combine differentiable physics with deep learning to represent solution manifolds rather than computing single solutions via optimization. In this way, trained models can interact with a physical environment using a large number of interaction parameters, and inference times are orders of magnitude faster than with classic optimization algorithms. Here the use of differentiable physics is key for a robust learning of the complex spaces of behavior encoded by the model PDEs.

In this context, we present *phiflow* (`https://github.com/tum-pbs/PhiFlow`), a fully differentiable Eulerian PDE framework that provides operators and solvers for a large class of PDEs with analytic gradients. By fully integrating the numerical solver into the training process, neural networks (NNs) can, e.g., learn to reduce numerical errors of PDE solvers, and to optimally control a physical system given an initial state and a target state. We show the capabilities of *phiflow* with a wide range of correction and control tasks for various advection-diffusion type PDEs, and demonstrate that long time frames can be handled via a specialized architecture and evaluation scheme that separates the learning of physical behavior for different time scales.

2 Differentiable PDE solvers

Let $\boldsymbol{u}(\boldsymbol{x}, t)$ be described by a PDE that can be explicitly solved forward in time, i.e. time and space derivatives do not mix. The PDE can then be written as

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathcal{P}\left(\boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}, \frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{x}^2}, ..., \boldsymbol{y}(t)\right) \tag{1}$$

where $\mathcal{P}$ models the physical behavior of the system and $\boldsymbol{y}(t)$ denotes any external factors that can influence the system. A classic solver can move the system forward in time via Euler steps:

$$\boldsymbol{u}(t_{i+1}) = \text{Solver}[\boldsymbol{u}(t_i), \boldsymbol{y}(t_i)] = \boldsymbol{u}(t_i) + \Delta t \cdot \mathcal{P}\left(\boldsymbol{u}(t_i), ..., \boldsymbol{y}(t_i)\right) \tag{2}$$

Workshop on Differentiable Vision, Graphics, and Physics in Machine Learning at NeurIPS 2020.

The square brackets indicate that Solver is a functional rather than a function, i.e. it takes full fields as input. Each step moves the system forward by a time increment Δt. Repeated execution produces a trajectory $u(t)$ that is a solution to the PDE.

When discretizing this formulation for time advancement directly it is not well-suited to solve optimization problems, since gradients can only be approximated by finite differencing in a regular forward solver. For high-dimensional or continuous systems, this method becomes computationally expensive because a full trajectory needs to be computed for each optimizable parameter. Differentiable solvers resolve this issue by solving the adjoint problem [12, 10] via analytic derivatives. The adjoint problem computes the same mathematical expressions while working with lower-dimensional vectors. A differentiable solver can efficiently compute the derivatives with respect to any of its inputs, i.e. $\partial \boldsymbol{u}(t_{i+1})/\partial \boldsymbol{u}(t_i)$ and $\partial \boldsymbol{u}(t_{i+1})/\partial \boldsymbol{y}(t_i)$. This allows for gradient-based optimization of inputs or control parameters of the simulation over an arbitrary number of time steps. The adjoint method is also used by most machine learning frameworks, where it is more commonly known as reverse mode differentiation [16, 3].

We make use of this analogy to realize *phiflow*, a differentiable PDE solver as a set of mathematical operations within a deep learning framework. We focus on Eulerian rather than Lagrangian methods since they are widely used for a large class of PDEs [14]. All solver operations are implemented in a differentiable manner, i.e. the automatic differentiation tools can chain the derivatives of these operations with built-in machine learning operations to build analytic derivatives for any combination of operations, thus enabling end-to-end training. This toolkit of operations enables the solver to handle a large class of PDEs, including the incompressible Navier-Stokes equations.

3 Learning solver interactions

Assuming the physical behavior $\mathcal{P}$ is described by a PDE as in Eq. (1), we add a force term $\boldsymbol{F}(t)$, which can be seen as a "correction" or "control" that allows the model to interact with the system:

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathcal{P}\left(\boldsymbol{u}, \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}, \frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{x}^2}, ...\right) + \boldsymbol{F}(t) \tag{3}$$

While the evolution of the complete state $\boldsymbol{u}$ is determined by the above equation, we allow some parts of $\boldsymbol{u}$ to be hidden for the forcing. This restriction reflects the fact that it is often not possible to observe the full state of a physical system. When considering a cloud of smoke, for example, the smoke density might be observable while the velocity field cannot be seen directly. Mathematically, we model this restriction by decomposing $\boldsymbol{u}$ into an observable part $\boldsymbol{o}$ and a hidden part $\boldsymbol{h}$ with $\boldsymbol{u} = \boldsymbol{o}(\boldsymbol{u}) \otimes \boldsymbol{h}(\boldsymbol{u})$. Here, $\otimes$ denotes the tensor product, adding all components of the states. The hidden part can include spatial regions of some fields as well as entire fields.

Using the above notation, we define the control task as follows. An initial observable state $\boldsymbol{o}_0$ of the PDE as well as a target state $\boldsymbol{o}^*$ are given. We are interested in a reconstructed trajectory $\boldsymbol{u}^r(t)$ that matches these states at t_0 and t_*, i.e. $\boldsymbol{o}_0 = \boldsymbol{o}(\boldsymbol{u}^r(t_0)), \boldsymbol{o}^* = \boldsymbol{o}(\boldsymbol{u}^r(t_*))$, and requires the least amount of effort over the whole time span. I.e., we aim for minimizing the forces to be applied in terms of their magnitude with:

$$L_{\boldsymbol{F}}[\boldsymbol{u}(t)] = \int_{t_0}^{t_*} |\boldsymbol{F}_{\boldsymbol{u}}(t)|^2 \, dt \tag{4}$$

Taking discrete time steps Δt, the reconstructed trajectory $\boldsymbol{u}^r$ is a sequence of $n = (t_* - t_0)/\Delta t$ states. This problem definition is portrayed in Fig. 1. An initial observation $\boldsymbol{o}_0$ and target observation $\boldsymbol{o}_*$ are given (a). The goal is to reconstruct a trajectory $\boldsymbol{u}^r$ that moves from $\boldsymbol{o}_0$ to $\boldsymbol{o}_*$ in the state space and requires as little force as possible, as shown in (b). The grey lines represent the unperturbed evolution of the physical system. The amount of applied force corresponds to how far the trajectory deviates from the natural evolution in this picture.

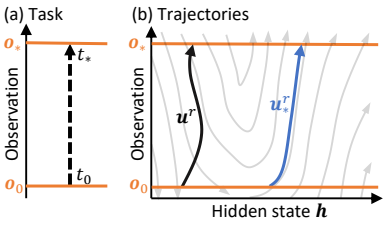

Figure 1: Possible trajectories.

When an observable dimension cannot be controlled directly, there may not exist any trajectory $\boldsymbol{u}(t)$ that matches both $\boldsymbol{o}_0$ and $\boldsymbol{o}^*$. This can stem from either physical constraints or numerical limitations.

In these cases, we settle for an approximation of $\boldsymbol{o}^*$. To measure the quality of the approximation of the target, we define an observation loss $L^*_{\boldsymbol{o}}$. The form of this loss can be chosen to fit the problem. For our experiments we use the filtered L_2 distance between target and reconstruction:

$$L^*_{\boldsymbol{o}}(\boldsymbol{u}(t_*)) = |B_r(\boldsymbol{o}^*) - B_r\left(\boldsymbol{o}(\boldsymbol{u}(t_*))\right)|^2 \tag{5}$$

where B_r denotes a spatial blur function with a fixed, problem-dependent radius $r \geq 0$. We combine Eqs. 4 and 5 into the objective loss function

$$L[u(t)] = \alpha \cdot L_{\boldsymbol{F}}[\boldsymbol{u}(t)] + \beta \cdot L^*_{\boldsymbol{o}}(u(t_*)), \tag{6}$$

with $\alpha, \beta > 0$. Since our solver is differentiable, L can be used directly to optimize a machine learning model such as a neural network that models $\boldsymbol{u}^r(t), \boldsymbol{o}_*, t \rightarrow \boldsymbol{F}(t)$ with weights $\boldsymbol{w}$. We call this network the control force estimator (CFE).

For a sequence of n frames, $L[\boldsymbol{u}(t)]$ depends on all n states of the trajectory $\boldsymbol{u}(t)$. Thus, for recurrent end-to-end training, n linked copies of the network need to be chained together. When inferring the force, this results in a CFE chain, shown in Fig. 2, that alternates between network and solver execution. When using a CFE chain, the complete sequence needs to be run forward and backward for each optimization step of the model. This is not only slow, it also means that gradients are passed through a potentially long chain of highly non-linear simulation steps. When the reconstruction u^r is close to an optimal trajectory, this is not a problem since the gradients $\Delta \boldsymbol{u}^r$ are small and the operations executed by the solver are differentiable by construction. The solver can therefore be locally approximated by a first-order polynomial and the gradients can be safely backpropagated. For large $\Delta \boldsymbol{u}^r$, such as at the beginning of training, this approximation breaks down, causing the gradients to become highly unstable while passing through the chain. In some cases below, we employ a second model, which predicts the observable state $\boldsymbol{o}^p\left((t_i + t_j)/2\right)$ given two observations. We refer to this model as the observation predictor (OP) [7].

Note that while some existing approaches rely on a continuous time formulation, e.g. for incorporating ODEs [3], we instead make use of a given time discretziation with a chosen temporal step size. While this requires storing the intermediate states of the simulated system, it allows for using numerical methods that are suitable to handle the specifics of a PDE under consideration. E.g., tailored time stepping schemes or specialized and efficient solvers can be integrated into the learning process in this way. E.g., we make use of these capabilities for the pressure calculation within a Navier-Stokes solver.

As this workshop paper can only provide a very brief summary of the different *phiflow* applications, the de-anonmyized version is this paper will refer to the *phiflow* source code and corresponding full papers.

4 Results

Here, we focus on *phiflow* applications in terms of two-dimensional fluid dynamic problems, which are highly challenging due to the complexities on the governing Navier-Stokes equations [2] for the velocity field $\boldsymbol{v}$,

$$\mathcal{P}(\boldsymbol{v}, \nabla \boldsymbol{v}) = -\boldsymbol{v} \cdot \nabla \boldsymbol{v} + \nu \nabla^2 \boldsymbol{v} + \nabla p, \tag{7}$$

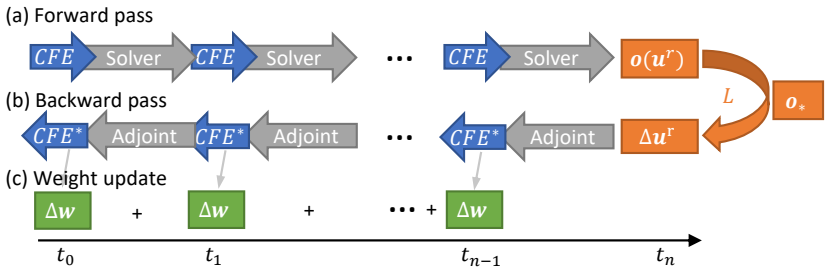

Figure 2: A chained force prediction network: (a) The forward pass reconstructs a trajectory by alternating between force estimation and solver execution. (b) For backpropagation, the adjoint problem is computed. (c) Weight updates are accumulated and applied to the model.

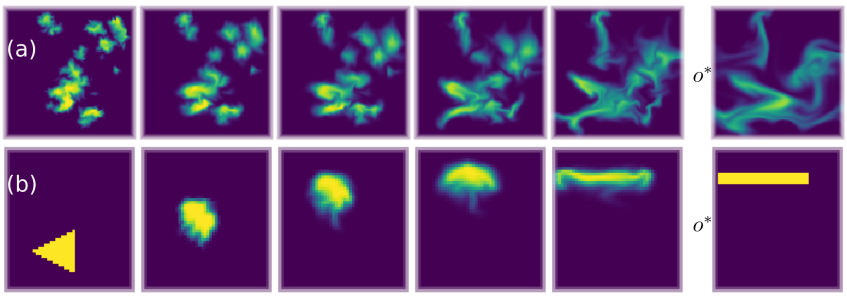

Figure 3: Example reconstructed trajectory from (a) the natural flow test set and (b) the shape test set. The initial state is shown on the far left, the target state $\boldsymbol{o}^*$ is shown on the right. The optimization goal for the NN is to reach the target state given the constraints of the physical model with forces for a wide range of randomized source and target states.

Table 1: A comparison of methods in terms of final cost for (a) a natural flow setup and (b) shape transitions from Fig. 3. The initial distribution is sampled randomly and evolved to the target state.

Execution	Loss	a) Force $L_{\boldsymbol{F}}$	a) Obs. $L^*_{\boldsymbol{o}}$	b) Force $L_{\boldsymbol{F}}$	b) Obs. $L^*_{\boldsymbol{o}}$
Regular	Supervised	243 ± 11	1.53 ± 0.23	n/a	n/a
Regular	*phiflow*	22.6 ± 1.1	0.64 ± 0.08	89 ± 6	0.331 ± 0.134
Refined	*phiflow*	11.7 ± 0.6	0.88 ± 0.11	75 ± 4	0.126 ± 0.010

subject to the hard constraints $\nabla \cdot \boldsymbol{v} = 0$ and $\nabla \times p = 0$, where p denotes pressure and ν the viscosity. In addition, we consider a passive density ρ which moves with the fluid via $\partial\rho/\partial t = -\boldsymbol{v} \cdot \nabla\rho$. We set $\boldsymbol{v}$ to be hidden and ρ to be observable and allow forces to be applied to all of $\boldsymbol{v}$.

Example sequences for the control task on 128×128 domains are shown in Fig. 3 and a quantitative evaluation, averaged over 100 examples, is given in Tab. 1. While all divide-and-conquer methods manage to approximate the target state well, there are considerable differences in the amount of force applied. The supervised technique, denoted as *regular*, exerts significantly more force than the differentiable solver based methods, resulting in jittering reconstructions. A prediction refinement scheme (denoted as *refined*) re-evaluates predictions over the course of a sequence. This version produces the smoothest transitions, converging to about half the loss of the regular, non-refined variant. For comparison, we run a classic optimization with hierarchical shooting that computes solutions for single cases, and find that it requires 1500 iterations to compute a control function that our trained model infers almost instantly. In the accompanying publications, we also demonstrate that more indirect forms of control of systems such as a Navier-Stokes environments are possible.

Additionally, combining differentiable PDE solvers and deep learning can be leveraged to reduce numerical errors, by omitting the OP network, and prediciton a correction for each step of a sequence via a CFE network. This is demonstrated for a 3D case of incompressible unsteady wake flow in 4. While a traditional, supervised version fares poorly and becomes unstable (not shown), the SOL_{16} version (trained with 16 steps of differentiable physics) achieves stable rollouts for several hundred time steps and successfully corrects the numerical inaccuracies of the coarse discretization. It improves the numerical accuracy of the source (SRC) simulation by more than 22% across a wide range of configurations. This case also highlights the gains in performance that can be achieved with our method: while the deep learning-based hybrid solver with SOL_{16} took 13.3s on average for 100 time steps, a CPU-based reference simulation required 913.2s. A speed-up of more than $68\times$.

5 Conclusions

We have introduced the *phiflow* framework with a summary of selected results. They show that deep learning models in conjunction with a differentiable physics solver can successfully predict the behavior of complex physical models and learn to control and correct them. We believe that learning differentiable physics has significant potential to provide physical intuition for a wide range of systems that understand and interact with the real world.

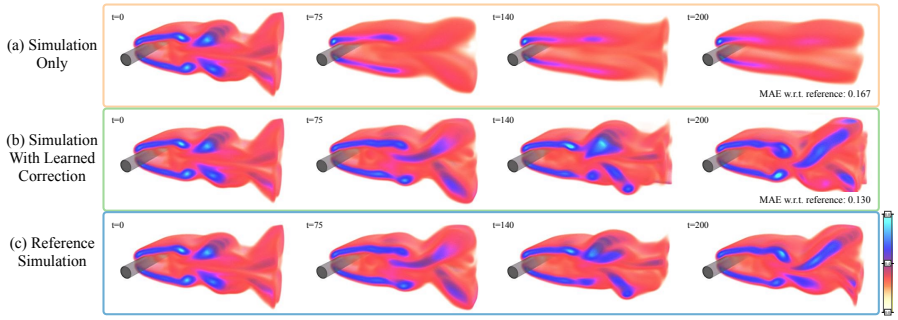

Figure 4: A 3D fluid problem, shown in terms of vorticity. From top to bottom: a) regular simulation, b) reference, c) regular simulation with learned corrector.

References

[1] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, 2016.

[2] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge University Press, 1967.

[3] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018.

[4] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, 2018.

[5] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, 2016.

[6] Nick Haber, Damian Mrowca, Li Fei-Fei, and Daniel LK Yamins. Learning to play with intrinsically-motivated self-aware agents. *arXiv:1802.07442*, 2018.

[7] Philipp Holl, Vladlen Koltun, and Nils Thuerey. Learning to control pdes with differentiable physics. *ICLR*, 2020.

[8] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

[10] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3), 2004.

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[12] Lev Semenovich Pontryagin. *Mathematical Theory of Optimal Processes*. John Wiley, 1962.

[13] Connor Schenck and Dieter Fox. SPNets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, 2018.

[14] Gordon D Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford University Press, 1985.

[15] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems*, 2018.

[16] Paul J Werbos. Backwards differentiation in AD and neural nets: Past links and new opportunities. In *Automatic Differentiation: Applications, Theory, and Implementations*, pages 15–34. Springer, 2006.