Differentiable Physics for Improving the Accuracy of Iterative PDE-Solvers with Neural Networks

Kiwon Um[1,2]
kiwon.um@telecom-paris.fr

Yun (Raymond) Fei[3]
yf2320@columbia.edu

Philipp Holl[1]
philipp.holl@tum.de

Robert Brand[1]
robert.brand@tum.de

Nils Thuerey[1]
nils.thuerey@tum.de

Abstract

Finding accurate solutions to partial differential equations (PDEs) is a crucial task for a wide range of fields in physics and engineering. We target the problem of reducing numerical errors of iterative PDE solvers and compare different learning approaches for finding complex correction functions. In particular, we highlight the performance of differentiable physics networks for a wide variety of PDEs, from non-linear advection-diffusion systems to three-dimensional Navier-Stokes flows.

1 Introduction

Numerical methods are prevalent in science to improve the understanding of our world [14, 13, 7]. We specifically target the numerical errors that arise in the discretization of PDEs [4, 2]. We show that, despite the lack of closed-form descriptions, discretization errors can be seen as functions with regular and repeating structures and, thus, can be learned by neural networks. Once trained, such a network can be evaluated locally to improve the solution of a PDE-solver, i.e., to reduce its numerical error.

We demonstrate that neural networks can be successfully trained if they can *interact* with the respective PDE solver during training. To achieve this, we leverage differentiable simulations [1, 16]. While incorporating physical models into deep learning approaches has been studied in various forms [10, 5, 3, 11], differentiable simulations allow a trained model to autonomously explore and experience the physical environment and receive directed feedback regarding its interactions throughout the solver iterations. We specifically target recurrent interactions of highly non-linear PDEs with deep neural networks. This combination bears particular promise: it improves generalizing capabilities of the trained models by letting the PDE-solver handle large-scale changes to the data distribution such

Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada.

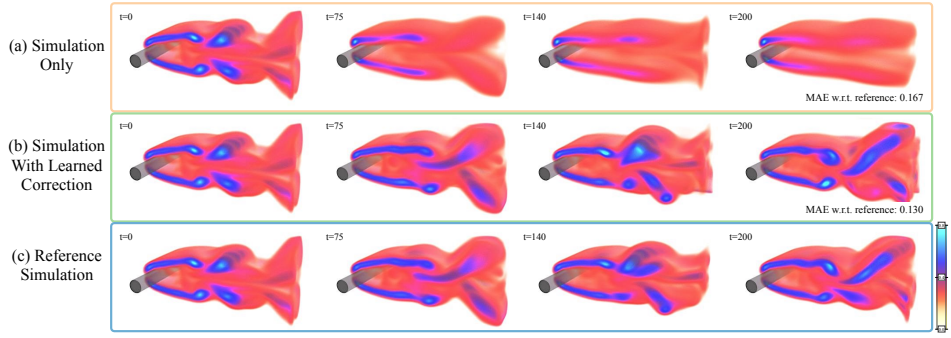


Figure 1: A 3D fluid problem, shown in terms of vorticity. From top to bottom: a) regular simulation, b) reference, c) regular simulation with learned corrector.

that the learned model can focus on localized structures not captured by the discretization. While physical models generalize very well, learned models often specialize in data distributions seen at training time. However, we will show that, by combining PDE-based solvers with a learned model, we can arrive at hybrid methods that yield improved accuracy while handling solution manifolds with significant amounts of varying physical behavior.

We show the advantages of training via differentiable physics for explicit and implicit solvers applied to a broad class of canonical PDEs. We showcase models trained with up to 128 steps of a differentiable simulator and apply our model to complex three-dimensional (3D) fluid flows (Fig. 1). Additionally, we present a detailed empirical study of different approaches for training neural networks in conjunction with iterative PDE-solvers for recurrent rollouts of several hundred time steps. On the side of implicit solvers, we consider the Poisson problem [9].

2 Learning to Reduce Numerical Errors

Numerical methods yield approximations of a smooth function $\boldsymbol{u}$ in a discrete setting and invariably introduce errors. These errors can be measured in terms of the deviation from the exact analytical solution. For discrete simulations of PDEs, they are typically expressed as a function of the truncation, $O(\Delta t^k)$. Higher-order methods, with large k, are preferable but difficult to arrive at in practice. For practical schemes, no closed-form expression exists for truncation errors, and the errors often grow exponentially as solutions are integrated over time. We investigate methods that solve a discretized PDE $\mathcal{P}$ by performing discrete time steps Δt. Each subsequent step can depend on any number of previous steps, $\boldsymbol{u}(\boldsymbol{x}, t+\Delta t) = \mathcal{P}(\boldsymbol{u}(\boldsymbol{x}, t), \boldsymbol{u}(\boldsymbol{x}, t-\Delta t), ...)$, where $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^d$ for the domain Ω in d dimensions, and $t \in \mathbb{R}^+$.

Problem Statement: We consider two different discrete versions of the same PDE $\mathcal{P}$, with $\mathcal{P}_R$ denoting a more accurate discretization with solutions $\mathbf{r} \in \mathscr{R}$ from the *reference manifold*, and an approximate version $\mathcal{P}_s$ with solutions $\boldsymbol{s} \in \mathscr{S}$ from the *source manifold*. We consider $\mathbf{r}$ and $\boldsymbol{s}$ to be states at a certain instance in time, i.e., they represent phase space points, and evolutions over time are given by a trajectory in each solution manifold. As we focus on the discrete setting, a solution over time consists of a *reference sequence* $\{\mathbf{r}_t, \mathbf{r}_{t+\Delta t}, \cdots, \mathbf{r}_{t+k\Delta t}\}$ in the solution manifold $\mathscr{R}$, and correspondingly, a more coarsely approximated *source sequence* $\{\boldsymbol{s}_t, \boldsymbol{s}_{t+\Delta t}, \cdots, \boldsymbol{s}_{t+k\Delta t}\}$ exists in the solution manifold $\mathscr{S}$. We also employ a mapping operator $\mathcal{T}$ that transforms a phase space point from one solution manifold to a suitable point in the other manifold, e.g., for the initial conditions of the sequences above, we typically choose $\boldsymbol{s}_t = \mathcal{T}\mathbf{r}_t$. In the simplest case, the projection $\mathcal{T}$ can be obtained via filtering and re-sampling operations.

By evaluating $\mathcal{P}_R$ for $\mathscr{R}$, we can compute the points of the phase space sequences, e.g., $\mathbf{r}_{t+\Delta t} = \mathcal{P}_R(\mathbf{r}_t)$ for an update scheme that only depends on time t. Without loss of generality, we assume a fixed Δt and denote a state $\mathbf{r}_{t+k\Delta t}$ after k steps of size Δt with $\mathbf{r}_{t+k}$. Due to the inherently different numerical approximations, $\mathcal{P}_s(\mathcal{T}\mathbf{r}_t) \neq \mathcal{T}\mathbf{r}_{t+1}$ for the vast majority of states. In chaotic systems, such differences typically grow exponentially over time until they saturate at the level of mean difference between solutions in the two manifolds. We use an L^2-norm in the following to quantify the deviations, i.e., $\mathcal{L}(\boldsymbol{s}_t, \mathcal{T}\mathbf{r}_t) = \|\boldsymbol{s}_t - \mathcal{T}\mathbf{r}_t\|_2$. Our learning goal is to arrive at a correction operator $\mathcal{C}(\boldsymbol{s})$ such that a solution to which the correction is applied has a lower error than an unmodified solution: $\mathcal{L}(\mathcal{P}_s(\mathcal{C}(\mathcal{T}\mathbf{r}_{t_0})), \mathcal{T}\mathbf{r}_{t_1}) < \mathcal{L}(\mathcal{P}_s(\mathcal{T}\mathbf{r}_{t_0}), \mathcal{T}\mathbf{r}_{t_1})$. The correction function $\mathcal{C}(\boldsymbol{s}|\theta)$ is represented as a deep neural network with weights θ and receives the state $\boldsymbol{s}$ to infer an additive correction field with the same dimension. To distinguish the original phase states $\boldsymbol{s}$ from corrected ones, we denote the latter with $\tilde{\boldsymbol{s}}$, and we use an exponential notation to indicate a recursive application of a function, i.e.,

$$\boldsymbol{s}_{t+n} = \mathcal{P}_s(\mathcal{P}_s(\cdots \mathcal{P}_s(\mathcal{T}\mathbf{r}_t)\cdots)) = \mathcal{P}_s^n(\mathcal{T}\mathbf{r}_t)\ . \tag{1}$$

Within this setting, any type of learning method naturally needs to compare states from the source domain with the reference domain in order to bridge the gap between the two solution manifolds. How the evolution in the source manifold at training time is computed, i.e., if and how the corrector interacts with the PDE, has a profound impact on the learning process and the achievable final accuracy. We distinguish three cases: no interaction (NON), a pre-computed form of interaction (PRE), and a tight coupling via a differentiable solver in the training loop (SOL). For each, a subscript n optionally denotes the number of steps over which the future evolution is recursively evaluated, e.g., SOL_n.

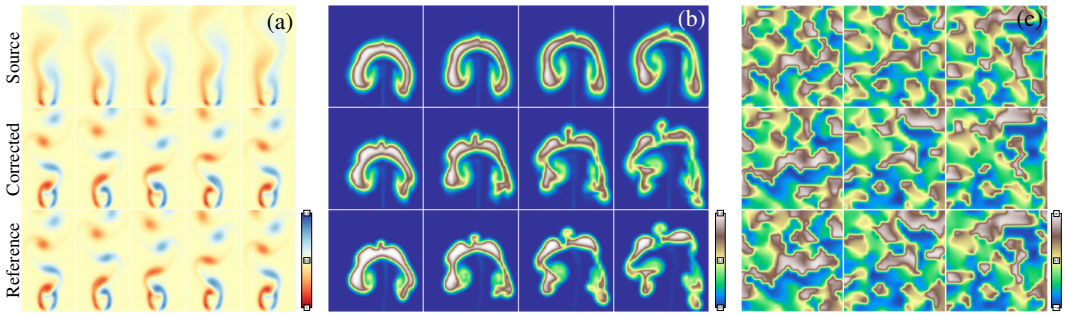


Figure 2: Our PDE scenarios cover a wide range of behavior including (a) vortex shedding, (b) complex buoyancy effects, and (c) advection-diffusion systems. Shown are different time steps (l.t.r.) in terms of vorticity for (a), transported density for (b), and angle of velocity direction for (c).

3 Experiments

We investigate a diverse set of constrained advection-diffusion models of which the general form is

$$\partial \boldsymbol{u}/\partial t = -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nu \nabla \cdot \nabla \boldsymbol{u} + \mathbf{g} \quad \text{subject to} \quad \boldsymbol{M}\boldsymbol{u} = 0, \tag{2}$$

where $\boldsymbol{u}$ is the velocity, ν denotes the diffusion coefficient (i.e., viscosity), and $\mathbf{g}$ denotes external forces. The constraint matrix $\boldsymbol{M}$ contains an additional set of equality constraints imposed on $\boldsymbol{u}$. In total, we target four scenarios: pure non-linear advection-diffusion (Burger's equation), two-dimensional Navier-Stokes flow, Navier-Stokes coupled with a second advection-diffusion equation for a buoyancy-driven flow, and a 3D Navier-Stokes case. Also, we discuss CG solvers in the context of differentiable operators below. The reference solutions from $\mathscr{R}$ are typically computed with the same numerical method using a finer discretization (4x in our setting, with effective resolutions of 128^2 and higher). For the SOL variants, we employ a differentiable PDE-solver that runs mini-batches of simulations and provides gradients for all operations of the solving process within the deep learning framework. For $n > 1$, i.e., PDE-based look-ahead at training time, the gradients are back-propagated through the solver $n-1$ times, and the difference w.r.t. a pre-computed reference solution is evaluated for all intermediate results.

The neural network component $F(\boldsymbol{s} \,|\, \theta)$ of the correction function is realized with a fully convolutional architecture. Our networks typically consist of 10 convolutional layers with 16 features each, interspresed with ReLU activation functions using kernel sizes of 3^d and 5^d. The networks parameters θ are optimized with a fixed number of steps with an ADAM optimizer [8] and a learning rate of 10^{-4}.

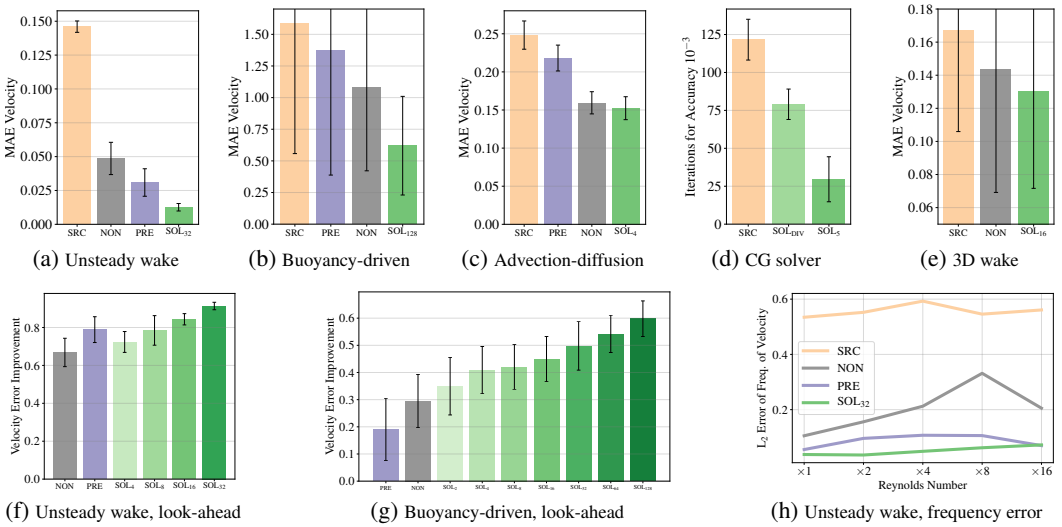
We quantify the performance of the trained models by computing the mean absolute error between a computed solution and the corresponding projected reference for n consecutive steps of a simulation. We report absolute error values for different models in comparison to an unmodified source trajectory from $\mathscr{S}$. Additionally, relative improvements are given w.r.t. the difference between unmodified source and reference solutions.

4 Results

Our experiments show that learned correction functions can achieve substantial gains in accuracy over a regular simulation. A visual overview of the different tests is given in Fig. 2, and a summary of our evaluation is provided in Fig. 3.

The first scenario includes an unsteady wake flow [12] in Fig. 3a), where the simplest method (NON) yields stable training and a model that already reduces the mean absolute error (MAE) from 0.146 for a regular simulation without correction (SRC) to an MAE of 0.049 when applying the learned correction. The pre-computed correction (PRE) improves on this behavior via its time regularization with an error of 0.031. A SOL_{32} model trained with a differentiable physics solver for 32 time steps in each iteration of ADAM yields a significantly lower error of 0.013. This means the numerical errors of the source simulation w.r.t. the reference were reduced by more than a factor of 10.

For the second scenario, buoyancy driven flows, the correction functions benefit from particularly long rollouts at training time in this scenario (Fig. 3b). Models with simple pre-computed or unaltered trajectories yield mean errors of 1.37 and 1.07 compared to an error of 1.59 for the source simulation,



(a) Unsteady wake (b) Buoyancy-driven (c) Advection-diffusion (d) CG solver (e) 3D wake

(f) Unsteady wake, look-ahead (g) Buoyancy-driven, look-ahead (h) Unsteady wake, frequency error

Figure 3: (a)-(e) Numerical approximation error w.r.t. reference solution for unaltered simulations (SRC) and with learned corrections. The models trained with differentiable physics and look-ahead achieve significant gains over the other models. (f,g) Relative improvement over varying look-ahead horizons. (h) A frequency-based evaluation for the unsteady wake flow scenario.

respectively. Instead, a model trained with differentiable physics with 128 steps (SOL_{128}) successfully reduces the error to 0.62, an improvement of more than 59% compared to the unmodified simulation.

We additionally evaluate our method for a pure advection-diffusion case with Burger's equation (Fig. 3c) similar to [3], and for supporting a conjugate gradient solver [6, 15] in terms of an initial guess (Fig. 3d). While the differentiable physics has less effect in the Burger's case due to randomized forcing being present, it succeeds in propagating boundary condition information for the initial guesses of the conjugate gradient solver.

Lastly, we investigate a 3D case of incompressible flow. The overall setup is similar to the unsteady wake flow in two dimensions outlined above, but the third dimension extends the axes of rotation in the fluid from one to three, yielding a very significant increase in complexity. As a result, the flow behind the cylindrical obstacle quickly becomes chaotic and forms partially turbulent eddies, as shown in Fig. 1. This scenario requires significantly larger models to learn a correction function, and the NON version does not manage to stabilize the flow consistently. Instead, the SOL_{16} version achieves stable rollouts for several hundred time steps and successfully corrects the numerical inaccuracies of the coarse discretization, improving the numerical accuracy of the source (SRC) simulation by more than 22% across a wide range of configurations (Fig. 3e).

This case also highlights the gains in performance that can be achieved with our method. Here, a simulation as shown in Fig. 1 involving the trained model took 13.3s on average for 100 time steps, whereas a CPU-based reference simulation required 913.2s. A speed-up of more than 68×. The graphs shown in Fig. 3 (f-h) additionally highlight the performance of models with increased look-ahead and provide an evaluation in terms of frequency content for the unsteady wake case.

5 Impact Statement

PDE-based models are very commonly used and can be applied to a wide range of applications, including weather and climate, epidemics, civil engineering, manufacturing processes, and medical applications. Our work has the potential to substantially improve how these PDEs are solved, by enabling a hybrid method in which an ANN learns to interact and improve the prediction of a traditional sovler. While our method could be used in the development of undesired harmful systems, our method shares this danger with the class of numerical methods in general.

References

[1] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 2017.

[2] V. I. Arnold. *Geometrical methods in the theory of ordinary differential equations*, volume 250. Springer Science & Business Media, 2012.

[3] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[4] S. Ghosal. An analysis of numerical errors in large-eddy simulations of turbulence. *Journal of Computational Physics*, 125(1):187–206, 1996.

[5] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15353–15363, 2019.

[6] M. R. Hestenes, E. Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[7] B. M. Johnston, P. R. Johnston, S. Corney, and D. Kilpatrick. Non-newtonian blood flow in human right coronary arteries: steady state simulations. *Journal of biomechanics*, 37(5):709–720, 2004.

[8] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs]*, Dec. 2014.

[9] J. Mathews and R. L. Walker. *Mathematical methods of physics*, volume 501. WA Benjamin New York, 1970.

[10] J. Morton, A. Jameson, M. J. Kochenderfer, and F. Witherden. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems*, 2018.

[11] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv:1808.04327*, 2018.

[12] B. Rajani, A. Kandasamy, and S. Majumdar. Numerical simulation of laminar flow past a circular cylinder. *Applied Mathematical Modelling*, 33(3):1228–1247, 2009.

[13] T. F. Stocker, D. Qin, G.-K. Plattner, M. Tignor, S. K. Allen, J. Boschung, A. Nauels, Y. Xia, V. Bex, P. M. Midgley, et al. Climate change 2013: The physical science basis. *Contribution of working group I to the fifth assessment report of the intergovernmental panel on climate change*, 1535, 2013.

[14] K. E. Taylor, R. J. Stouffer, and G. A. Meehl. An overview of cmip5 and the experiment design. *Bulletin of the American Meteorological Society*, 93(4):485–498, 2012.

[15] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of Machine Learning Research*, pages 3424–3433, 2017.

[16] M. Toussaint, K. Allen, K. Smith, and J. B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems*, 2018.