MASS CONSERVATIVE NEURAL NETWORKS

Inauguraldissertation

zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

Fabricio Arend Torres

2024

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät auf Antrag von

Prof. Dr. Volker Roth, Erstbetreuer

Prof. Dr. Thomas Vetter, Zweitbetreuer

Prof. Dr. Maarten Valentijn de Hoop, externer Experte

Basel, den 30. April 2024

Prof. Dr. Marcel Mayor, Dekan

To my family.

ABSTRACT

Neural networks have established themselves as a powerful tool for extracting insights from vast amounts of data. However, with increasing use of deep learning in the natural sciences, there is also an increasing demand to incorporate domain-specific expert knowledge. In many cases, this knowledge comes in the form of constraints expressed as partial differential equations (PDE). In addition to possibly improving generalization capabilities, enforcing such constraints would ensure predictions that are consistent with available domain knowledge. In this thesis, we focus on enforcing the physical law of mass conservation in neural networks, with the aim of modeling densities and velocities of compressible fluids. Specifically, we enforce the continuity equation, a PDE describing mass conservation in its local and differential form. We focus on models in continuous space and time.

In our first contribution, we weakly enforce the continuity equation by minimizing a PDE penalty on the so-called collocation points. Specifically, we provide an extension to physics-informed neural networks (PINNs). Motivated by the microscopic perspective of a fluid density, we propose to select collocation points by sampling particles from the (normalized) fluid density with dynamic Monte Carlo methods. This mesh-free and adaptive sampling method improves the sample efficiency for enforcing the continuity equation and other density-based advection-diffusion PDEs, which we demonstrate through various experiments.

In our second contribution, we propose Lagrangian Flow Networks (LFlows), a framework for constructing neural networks that adhere to the continuity equation by construction. We do so by leveraging insights from classical theory on Lagrangian flows, which allow us to model physically consistent densities and velocities with time-conditioned diffeomorphisms, i. e. conditional Normalizing Flows. This approach not only offers high predictive accuracy in density modeling tasks, but also proves computationally efficient. We showcase LFlows in both 2D and 3D scenarios, and apply it to the real-world application of bird migration modeling.

In summary, we study different approaches for incorporating PDEs, particularly the continuity equation, into neural networks for modeling compressible fluids. The resulting methods ensure physical consistency of the predictions while maintaining computational efficiency.

ACKNOWLEDGEMENTS

When starting my journey towards a Ph.D., I was already well aware that research happens in collaboration rather than isolation. However, with nearly two years of home office due to COVID measures, I have learned that this encompasses not only planned and structured meetings. It turned out that in the right environment seemingly small daily interactions such as shared coffee and lunch breaks, office discussions about code, and general small-talk were nearly as important. They contributed to the formulation and refinement of ideas, and enabled interactions that finally culminated in this thesis. Before acknowledging specific individuals, I would thus like to thank everyone at the Department of Computer Science and Mathematics at the University of Basel. The cooperative and collaborative work environment of the last few years is one that I aim to maintain throughout my future.

Throughout my Ph.D. studies, I benefited from the freedom to explore vastly different directions. For this opportunity as well as for the valuable feedback and knowledge gained in the past few years, I would like to thank my supervisor Volker Roth. It was fascinating to get a peek into some very different application areas of Machine Learning while being part of the BMDA group. I always enjoyed our group meetings, which provided a space for discussion of sometimes admittedly abstract concepts and helped shape my overall understanding of Machine Learning.

Next, I would like to thank my second supervisor, Thomas Vetter, for his advice and insight in my Committee meetings and during the reading groups. In addition, his "Pattern Recognition" class in my Bachelor's degree was the one that actually piqued my interest in the field of Machine Learning. I always appreciated the joint coffee breaks, hikes, and reading groups with the GRAVIS group.

I would like to also acknowledge my external reviewer Maarten de Hoop for his interest in my work and his time and effort spent on reviewing this thesis.

Furthermore, I am thankful to Ivan Dokmanić for the valuable discussions on Machine Learning. His input pointed me towards physics-informed neural networks and other interesting works related to physics-informed Machine Learning.

I would be remiss not to mention Christine Alewell, who enabled me a part-time employment in her welcoming research group. This provided

me not only with a lot of flexibility in the last steps of my thesis, but also with an interesting Machine Learning project that I am very much looking forward to.

I am deeply grateful to all current and former members of the BMDA group: Sonali Parbhoo, Mario Wieser, Sebastian Keller, Aleksander Wieczorek, Damian Murezzan, Maxim Samarin, Daniel Hauke, Vitali Nesterov, Stephan Unter, Monika Nagy-Huber, Jonathan Aellen, and Marcello Negri. From teaching and co-supervising me in my undergraduate theses to jointly correcting "Scientific Computing" exams and finally aiming for conference deadlines in my Ph.D., you all accompanied me on my journey and I am glad to have had you as my colleagues. Special thanks go to Marcello Negri for providing me feedback on parts of my thesis, and for joining me on the many late night crunches for our publications.

Likewise, I would like to thank the many former and current Ph.D. students, researchers, and staff of our department for enjoyable lunches, department-wide activities, and interesting research retreats. Special thanks go (in no particular order) to Dana Rahbani, Xolisile Thusini, Dennis Madsen, Thomas Sutter, Imant Daunhawer, Marcel Lüthi, Marco Vogt, Vinith Kishore, Marco Inversi, Valentin Debarnot, Jonas Linkerhägner, AmirEhsan Khorashadizadeh, Enea Compagnoni, Simon Dold, Enrico Giudice, Bas Kin, as well as all other members of the former GRAVIS, SADA and the Optimization of ML Systems groups.

I am grateful towards my family and relatives throughout the world - from Saarland to Costa Rica. Although we meet eachother rarely, you are always with me.

Insbesondere möchte ich meinen Eltern Elena und Peter, meiner Partnerin Vivian, und meinem Bruder Sergio sowie seiner Partnerin Grace danken, ohne die Ich diesen Weg nicht hätte bestreiten können. Danke, dass ich stets auf euch zählen konnte und ihr mir immer Rat zur Hand hattet. Zusätzlich möchte ich meiner Oma Wilfriede dafür danken, dass sie mich stets unterstützt hat.

Zuletzt geht mein tiefster Dank an meine Freunde. Ob Bouldern, Billiard, Musik oder einfach ein Bier im Bistro - Ihr habt mir immer einen Rückzugsort gegeben, an dem ich einfach ich selbst sein konnte und mir die Möglichkeit gegeben, meine Arbeit für eine Weile zu vergessen. Mein Dank geht ausdrücklich (in beliebiger Reihenfolge) an Daniel Forat, Tim Lüber, Lars Lucas/Weiser, Franziska Weiser, Adrian Greiner, Felix Sattler, Denis Lüber, Bianca Schindler, Tom Spitz, Daniel Trüby, Alain Studer, Pasqual Karasch und Christian Schadt.

CONTENTS

NOTATION

We base our mathematical notation, as well as this overview, on the template freely provided by the authors of Goodfellow et al. (2016)[1].

NUMBERS AND ARRAYS

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\mathrm{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{A}$	A vector-valued random variable

SETS

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0,1\}$	The set containing 0 and 1
$\{0,1,\dots,n\}$	The set of all integers between 0 and n
$[a,b]$	The real interval including a and b
$(a,b]$	The real interval excluding a but including b
$\mathbb{A}\backslash\mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$

1 https://github.com/goodfeli/dlbook_notation, Accessed 01.02.24.

INDEXING

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
a_i	Element i of the random vector $\mathbf{a}$

LINEAR ALGEBRA OPERATIONS

$\boldsymbol{A}^\top$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\det(\boldsymbol{A})$	Determinant of $\boldsymbol{A}$

CALCULUS

$\frac{dy}{dx}$ or $d_x y$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$ or $\partial_x y$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla \cdot \boldsymbol{b} = \nabla_{\boldsymbol{x}} \cdot \boldsymbol{b}$	Divergence of vectorfield $\boldsymbol{b} : \mathbb{R}^d \mapsto \mathbb{R}^d$ w.r.t. $\boldsymbol{x} \in \mathbb{R}^d$ given by $\sum_{i=0}^{d} \frac{\partial \boldsymbol{b}_i}{\partial x_i}$
$\Delta_{\boldsymbol{x}} y = \nabla_{\boldsymbol{x}} \cdot \nabla_{\boldsymbol{x}} y$	Laplacian of y with respect to $\boldsymbol{x}$
$\frac{\partial f}{\partial \boldsymbol{x}}(\boldsymbol{y})$ or $Jf_{\boldsymbol{x}}(\boldsymbol{y})$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ evaluated at $\boldsymbol{y}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x}\sim p}[f(x)]$ or $\mathbb{E}[f(x)]$	Expectation of $f(x)$ with respect to $p(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $p(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $p(\mathrm{x})$
$D_{\mathrm{KL}}(P\|Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

FUNCTIONS

$f : \mathbb{A} \mapsto \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. We sometimes write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation.
$\ln x$	Natural logarithm of x
$\sigma(\boldsymbol{x})$	An elementwise function, s.t. $[\sigma(\boldsymbol{x})]_i = \sigma(x_i)$
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
$\mathrm{sgn}(x)$	sign of x
$\exp(x)$	e^x

TOOLS USED

For grammar and spelling, the writing assistant tool "Writeful for Overleaf"[2] was used for sentence-level corrections. Only "Writefull's language check" was used. None of the advanced widgets were used.

2 https://www.writefull.com/; Chrome Extension Version 2.2.0; Accessed March 2024.

1 INTRODUCTION

Machine Learning has become known for its ability to extract and act on information obtained from previously insurmountable amounts of data and its transformative effect on many scientific fields. Taking physical sciences as an example, deep learning enables simulations of molecules (Schütt et al., 2018) or weather phenomena (Pathak et al., 2022) at significantly reduced costs compared to traditional methods. This improvement in speed is possible by training a deep neural network on simulated data and thus encouraging the network to learn the underlying dynamics, providing a fast surrogate to the expensive simulators. In other words, the domain-specific knowledge in the form of a physical model is indirectly transferred to the neural network through massive amounts of data. While weather and molecule simulations are just two examples of many, they already illustrate the effect and potential of deep learning in the natural sciences.

But the interaction between different fields is, of course, not unidirectional. Although not as prominently discussed and more subtle, different scientific disciplines had a long-lasting impact on Machine Learning itself. Many core tools and concepts in Machine Learning draw inspiration and conceptual ideas from areas such as neuroscience, information theory, dynamical systems, and physics. Notable examples of physics-inspired concepts include energy-based models (LeCun et al., 2006; Smolensky et al., 1986), mean-field theory for variational inference (Zhang et al., 2018), or current breakthroughs in image generation based on diffusion processes in statistical physics (Rombach et al., 2022; Sohl-Dickstein et al., 2015). The underlying idea is that mirroring the constraints and approximations of, for example, physical models in the design and training of a Machine Learning model may be beneficial for a wider class of problems.

A natural combination of these development is to not only tackle problems from the physical sciences with Machine Learning, but to also leverage the vast expert knowledge culminated in these domains by enforcing it into the models. This leads us to the field of physics-informed Machine Learning (PI-ML)(Karniadakis et al., 2021). PI-ML focuses on modeling sparse and noisy sensor data while explicitly enforcing problem-specific physical constraints. That is, it combines a physical model available in the form of partial differential equations (PDEs) with sparse and noisy data, similar to a classical data assimilation (Asch et al., 2016) setting. A notable difference between PI-ML and data assimilation is, however, that the latter mostly assumes a reasonably correct and complete physical forward model (Geer, 2021). If the physical model is incomplete and has more degrees of freedom, there is a greater need for the generalization capabilities of Machine Learning. Physics-informed Machine Learning thus mostly shines in such ill-posed settings with partial knowledge about the physical model (Karniadakis et al., 2021). Finally, while the previously mentioned surrogate models can learn physical laws from large amounts of dense observations, such data is rarely available in realistic settings.

Within this thesis, we contribute to a subset of the very broad field of PI-ML; we constrain neural networks to predict physically consistent density movements. Specifically, we enforce the PDE that governs the law of mass conservation as a physical constraint.

1.1 PROBLEM SETTING AND MOTIVATION

The setting of this thesis was motivated by some of the overarching challenges present in modeling large-scale movements of airborne animals, where only sparse observations density and velocity are available. This section provides a brief background on this setting, leading to the general objective of this thesis.

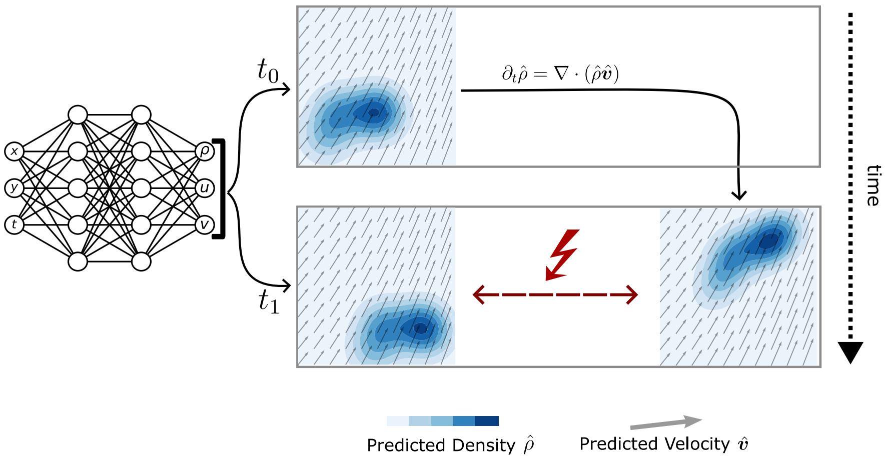
Largte-scale measurements of airborne animals are a product of recent developments in the field of radar aeroecology (Chilson et al., 2017). The field of radar aeroecology generally concerns itself with extracting information about airborne animals such as birds (Dokter et al., 2011; Eastwood, 1967; Gasteren et al., 2008), bats (Horn & Kunz, 2008; Stepanian et al., 2019; Williams et al., 1973), and insects (Rainey, 1955) from (weather) radar scans. With modern post-processing pipelines, it is possible to then estimate the average densities of large flocks (or swarms) from the reflectivity of radar scans, filtering out scatter and rain (Dokter et al., 2011; Stepanian et al., 2019). In addition, even information on the average velocity can be inferred from Doppler radars (Chilson et al., 2017). Naturally, with such large-scale measurements available, there has been growing interest in modeling the spatiotemporal movements of e.g. migratory birds (Nussbaumer et al., 2019; 2021). These models hold the potential to offer deeper insight into the behavior of migrants. Initial approaches to modeling such data were mainly based on Gaussian processes (Nussbaumer et al., 2019) or tree-based methods (Van Doren & Horton, 2018). In these works, the predictions of densities and velocities are treated as completely separate regression problems, ignoring the velocity (Van Doren & Horton, 2018) or modeling it separately (Angell & Sheldon, 2018; Nussbaumer et al., 2019).

When the velocity and density fields are modeled separately, there is, however, no guarantee that they are physically consistent with each other. The predicted density might change differently over time than what is suggested by the predicted velocity (e.g. move in a different direction), as illustrated in Fig. 1.1. A lack of physical consistency in a model is a fundamental problem for downstream interpretations, since there may be two possibly disagreeing explanations. Furthermore, if only sparse measurements are available and these measurements also suffer from noise and bias, physical inconsistency of the model is all but guaranteed. Consequently, it is of interest to ensure density and velocity predictions that agree with each other.

This intuitive notion of consistency between density movements and velocity has an underlying physical principle, namely the physical law of mass conservation. This law of mass conservation can be expressed in its local and differential form as a partial differential equation, namely, the continuity equation. Thus, to guarantee "physically consistent" densities and velocities, the continuity equation is to be enforced in the ML model, which is in our case a neural network. However, towards the beginning stages of this thesis, there was very limited work on actually enforcing mass conservation in neural networks without relying on any discretization, with the exception of general-purpose frameworks such as physics-informed neural networks (Raissi et al., 2019). This led us to the pursuit of a more specialized modeling framework for mass conservative neural networks.

Problem Setting. The general goal of this thesis is to develop neural networks that jointly model (physical) densities $\rho : [t_0, T] \times \Omega \mapsto \mathbb{R}_{\geq 0}$ with $\int_\Omega \rho(\boldsymbol{x}, t)\, \mathrm{d}\boldsymbol{x} < \infty$ and velocities $\boldsymbol{v} : [t_0, T] \times \Omega \mapsto \mathbb{R}^d$, with spatial domain $\Omega \subseteq \mathbb{R}^d$ and time $t \in [t_0, T] \subset \mathbb{R}_{\geq 0}$. Specifically, these are to be learned from noisy and sparsely distributed observations of the density $\{\rho_i\}_{i=1}^n$ and optionally the velocity $\{\boldsymbol{v}_j\}_{j=1}^m$. Finally, we assume that the phenomena we model follow the physical law of mass conservation and consequently aim to enforce this constraint in the model. That is, the velocity $\boldsymbol{v}(t, \boldsymbol{x})$ has to describe the evolution of the density $\rho(t, \boldsymbol{x})$ by fulfilling the partial differential equation

$$\partial_t \rho = -\nabla \cdot (\rho \boldsymbol{v}). \tag{1.1}$$



t_0
t_1
x
y
t
ρ
u
v
$\partial_t\hat{\rho} = \nabla \cdot (\hat{\rho}\hat{\boldsymbol{v}})$
time
Predicted Density $\hat{\rho}$
Predicted Velocity $\hat{\boldsymbol{v}}$

FIGURE 1.1: Visualization of the general problem setting. The left column indicates neural network predictions of the density and velocity at t_0 and t_1. The right column shows the density at t_1 that would fulfill the continuity equation given $\hat{\rho}(t_0, \boldsymbol{x})$ and $\hat{\boldsymbol{v}}(t, \boldsymbol{x})$. Intuitively, the density movements from t_0 to t_1 must match with the movements indicated by the velocity.

This PDE is commonly known as the *continuity equation* (CE) and corresponds to the law of mass conservation in its local and differential form. In summary, the objective is to develop neural network-based models that enforce the continuity equation in Eq. (1.1) for predictions of ρ and $\boldsymbol{v}$.

1.2 CONTRIBUTION

With the objective of mass conservative neural networks introduced, we provide a short overview of our contributions. The work presented in this thesis is split into two parts.

In the first contribution covered by Chapter 3, we focus on the existing framework of physics-informed neural networks (PINNs) to enforce mass conservation. Typically, PINNs weakly enforce constraints by minimizing a PDE-penalty at randomly chosen locations within a fixed space and time region. These locations are referred to collocation points. If, however, the boundaries of the problem are not known or the target density only occupies a small subset of the domain, naive sampling schemes will provide collocation points that are mostly within low-density regions. When using few collocation points, this leads to difficulties in enforcing PDEs such as the continuity equation with few collocation points, especially in higher dimensions. As these difficulties are not restricted to the continuity equation, we approach this problem in a more general setting. In our work Arend Torres et al. (2022) we consider time-dependent PDEs in which the variable of interest can be interpreted as a density. We propose to sample collocation points based on the normalized predicted density, which can be viewed as uniformly sampling particle positions. The resulting particle-density PINNs (pdPINNs) overcome the aforementioned limitations and recover a often claimed, but in practice arguably not fulfilled, mesh-free property of PINNs.

In our second contribution, which is covered by Chapter 4, we develop a neural network architecture with density and velocity predictions that satisfy the continuity equation by construction. The resulting networks are called *Lagrangian Flow Networks* (LFlows), and the chapter closely follows our work presented in Torres et al. (2024). As the name suggests, LFlows leverage a Lagrangian viewpoint of the problem to (implicitely) model evolving densities of parcels. Specifically, they make use of classical theory on Lagrangian flows, which links the continuity equation to time-dependent diffeomorphisms and the corresponding pushforward of an initial density. In practice, the implementation of LFlows is based on conditional Normalizing Flows, which are commonly used to learn probability densities in Machine Learning. The physical density is computed by rescaling the (conditional) probability density of a time-conditioned Normalizing Flow. In addition to the density, we provide an analytical expression for the velocity that is always consistent with predicted density changes.

1.3 LIST OF PUBLICATIONS

This thesis is based on the following papers and some additional unpublished work.

- *Lagrangian Flow Networks for Conservation Laws.* (Spotlight)

  Fabricio Arend Torres, Marcello Massimo Negri, Marco Inversi, Jonathan Aellen, and Volker Roth.

  The Twelfth International Conference on Learning Representations (ICLR), 2024.

- *Mesh-free Eulerian Physics-Informed Neural Networks*

  Fabricio Arend Torres, Marcello Massimo Negri, Monika Nagy-Huber, Maxim Samarin and Volker Roth.

  arXiv preprint, 2022.

In addition, the subsequent publications resulted from work during the PhD, but are not covered within the content of this thesis.

- *Conditional Matrix Flows for Gaussian Graphical Models.*

  Marcello Massimo Negri, Fabricio Arend Torres and Volker Roth.

  Advances in Neural Information Processing Systems 36, 2024.

- *Learning Invariances with Generalised Input-convex Neural Networks.*

  Vitali Nesterov, Fabricio Arend Torres, Monika Nagy-Huber, Maxim Samarin and Volker Roth.

  arXiv preprint, 2022.

- *Learning Extremal Representations with Deep Archetypal Analysis.*

  Sebastian Mathias Keller, Maxim Samarin, Fabricio Arend Torres, Mario Wieser and Volker Roth.

  International Journal of Computer Vision, Volume 129, pages 805–820, 2021.

Finally, the use of bijective neural networks in multiple of the aforementioned works resulted in a Python library for conditional Normalizing Flows.

- *FlowConductor: (Conditional) Normalizing Flows and Bijective Layers for PyTorch.*

  Fabricio Arend Torres, Marcello Massimo Negri, Jonathan Aellen.

  https://github.com/FabricioArendTorres/FlowConductor

2 BACKGROUND

In this chapter, we provide the necessary background for following the thesis. We begin with Section 2.1, offering a high-level overview of Machine Learning, regularization, and physics-informed ML as a specialized form of regularization. Following this, Section 2.2 introduces the continuity equation from different perspectives, covering an Eulerian, Lagrangian, and a low-level particle view. Section 2.3 continues with deep learning concepts that we build upon, ranging from invertible neural networks to neural ordinary differential equations. Building upon these concepts, in Section 2.4 we discuss Normalizing Flows. Normalizing Flows are crucial for the proposed Lagrangian Flow Networks, which we present in Chapter 4. Finally, Section 2.5 addresses physics-informed neural networks (PINNs) for enforcing physical constraints through penalties. These form the basis for the particle-density PINNs in Chapter 3.

2.1 MACHINE LEARNING

The field of Machine Learning is centered around the construction of models that approximate a largely unknown process based on limited and noisy observations. The heavy reliance on data sets it apart from models commonly employed in other scientific domains, as the process in question is typically either partially unknown or too complex to be explicitly modeled from first principles.

Based on the description by Vladimir Vapnik (Vapnik, 1999), (supervised) learning problems encompass three key components. Initially, we acquire random observations denoted as $\boldsymbol{x}$ from a generator (G) that characterizes the data collection process. These observations are independently sampled from a fixed but unknown probability distribution $p(\boldsymbol{x})$. A supervisor (S) then returns an output value or vector $\boldsymbol{y}$ to every input vector $\boldsymbol{x}$ according to a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ that is fixed and also unknown. This supervisor encapsulates the process we aim to approximate. The third and final element is a learning machine (LM), which employs a set of parameterized functions denoted as $f(\boldsymbol{x};\boldsymbol{\theta})$, with the parameters $\boldsymbol{\theta}$ belonging to a set $\boldsymbol{\Theta}$. The learning problem then consists of finding a function or model $f(\boldsymbol{x};\boldsymbol{\theta})$

that approximates the outputs provided by the supervisor by estimating $p(\boldsymbol{y}|\boldsymbol{x})$.

For selecting a function $f(\boldsymbol{x};\boldsymbol{\theta})$ the LM is restricted to a training data set of n independent observations $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=0}^{n}$ drawn from $p(\boldsymbol{x},\boldsymbol{y}) = p(\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{x})$. We denote the rowwise concatenation of the training observations with the matrices $\boldsymbol{X}_{train} = [\boldsymbol{x}_0^t, \boldsymbol{x}_1^t, \ldots, \boldsymbol{x}_n^t]^t$ and $\boldsymbol{Y}_{train} = [\boldsymbol{y}_0^t, \boldsymbol{y}_1^t, \ldots, \boldsymbol{y}_n^t]^t$. The parameters $\boldsymbol{\theta}$ are then estimated by minimizing a loss function $L(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{Y})$ - the empirical risk - on the observed training set:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{Y}) \tag{2.1}$$

The challenge in Machine Learning is now to design function spaces, optimization methods, and loss functions such that minimization of the empirical loss in Eq. (2.1) also leads to a small expected loss on unseen test data while remaining computationally feasible.

2.1.1 *Regularization*

Data in the real world often has limitations, such as being limited in quantity, containing noise, or exhibiting bias. These issues pose significant challenges in Machine Learning, where simply using models to interpolate training data to an arbitrary extent is inadequate, as it leads to overfitting. To address this regularization is essential, helping models perform well not only on existing training data, but also on new unseen data. Modern deep learning is not exempt from this. For example, the generalization capabilities of deep learning are often attributed to the implicit regularization provided by stochastic gradient descent methods (Gunasekar et al., 2018). More explicit regularization methods, such as dropout (Hinton et al., 2012) or spectral normalization (Miyato et al., 2018), are not only an active field of research, but are also highly relevant in practice (Goodfellow et al., 2016; Kukačka et al., 2017). In addition to improving generalization capabilities, however, regularization also serves as a means of incorporating more general prior knowledge. For instance, some predictions might be implausible to domain experts, and should thus be discouraged. Generative models of molecules, for example, often rely on latent representations that are invariant to a range of transformations (Gebauer et al., 2019; Nesterov et al., 2020), as they should not affect the probability of generating a molecule. If these constraints are in the form of partial differential equations that describe physical laws, this is commonly referred to as Physics-informed Machine Learning.

Physics-informed Machine Learning. Physics-informed Machine Learning (PI-ML) is a subfield that combines the expressive power of modern ML methods with physical constraints that serve as meaningful regularizers. The applications of PI-ML range widely, from those directly related to physical systems (Greydanus et al., 2019; Raissi et al., 2019) to more general domains such as generative models for videos (J. Li et al., 2023), or learning cellular dynamics of single-cell RNA sequencing (Tong et al., 2020). PI-ML can be mainly set apart from more general regularization methods by the form in which physical prior knowledge is available, namely partial or ordinary differential equations (PDEs and ODEs respectively). Differential equations are capable of describing many phenomena and laws in physics, leading to complex dynamical systems. Consequently, they provide a highly flexible and well-studied framework for constraining functions. In addition, the concrete mathematical formulation is highly compatible with modern differentiable Machine Learning models. This puts the physics domain in contrast with domains such as medicine, where it can be very difficult to translate expert knowledge into a form amenable to modern Machine Learning. Nevertheless, enforcing general PDEs in Machine Learning models are still a field of active research with many remaining challenges (see e.g. Karniadakis et al. (2021)).

Regularization as Constrained Optimization. In context of this thesis, we focus on the physics-based regularization of $f(x;\theta)$ that can be expressed as a constraint $c(\boldsymbol{\theta}) = \mathbf{0}$ with a differentiable function $c : \boldsymbol{\theta} \mapsto \mathbb{R}^{\dim}(\boldsymbol{c})$. This effectively turns Eq. (2.1) into the constrained optimization problem

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}\in\Theta} L(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{Y}) \\ &\text{subject to } \boldsymbol{c}(\boldsymbol{\theta}^*) = \mathbf{0}. \end{aligned} \tag{2.2}$$

A simple method to (weakly) enforce such constraints is a penalty-based approach. The constrained optimization problem is turned into an unconstrained problem by introducing an additional (weighted) penalty term to the loss objective:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}\in\Theta} L(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{Y}) + \boldsymbol{\lambda}^\top \boldsymbol{m}(|\boldsymbol{c}(\boldsymbol{\theta})|), \tag{2.3}$$

where $\boldsymbol{\lambda} \in \mathbb{R}_{\geq 0}^{\dim(\boldsymbol{c})}$ and $\boldsymbol{m}$ is an elementwise monotone transformation, such as the square. For a sufficiently high λ, this objective will enforce the given constraints without requiring any constraint-specific parameterization or projection. Due to its simplicity, it is a particularly popular approach in Deep Learning (Márquez-Neila et al., 2017). Physics-informed neural networks (Dissanayake & Phan-Thien, 1994; Raissi et al., 2017) make use of penalty based regularization for enforcing PDEs. In our first contribution *Mesh-Free Eulerian Physics-Informed Neural Networks* (Arend Torres et al., 2022) we also follow this approach for enforcing physics constraints.

While the penalty-based approach is simple to implement, it has some severe limitations. The constraints are only weakly enforced with a loss that is traded off with a data loss. Furthermore, additional hyperparameters must be selected and tuned. As the loss weighted by λ is traded off with the empirical loss, it has a strong effect on the model and its generalization. A more direct and strict approach to such constrained optimization is to directly constrain the search space of functions

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\Theta_c} L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) \qquad \Theta_c = \{\boldsymbol{\theta} \in \Theta : \boldsymbol{c}(\boldsymbol{\theta}) = \boldsymbol{0}\}. \tag{2.4}$$

The difficulty of this approach lies in finding a parameterization that always satisfies the constraints, or at least a projection onto such a parameterization. Unlike the penalty-based approach, this often requires specialized model architectures. We refer to Section 2.2 of Sharma et al. (2023) for some examples on enforcing physical constraints via specialized neural network architectures. In our second contribution *Lagrangian Flow Networks for Conservation Laws* (Torres et al., 2024) we follow such an approach. We present an architecture that directly parameterizes functions that fulfill the constraint of interest, namely the continuity equation.

2.2 MASS CONSERVATION AND THE CONTINUITY EQUATION

The laws describing the evolution of classical fluid flows are totally defined by the conservation of mass, momentum and energy (Hirsch, 2007) and can be formally expressed in terms of so-called conservation equations. The introductory text book by Hirsch (2007) describes the conservation law for any quantity as follows:

> *'The variation of the total amount of a quantity [. . .] inside a given domain is equal to the balance between the amount of that quantity entering and leaving the considered domain, plus the contributions from eventual [sinks and] sources generating that quantity.'* (Hirsch, 2007, p.29)

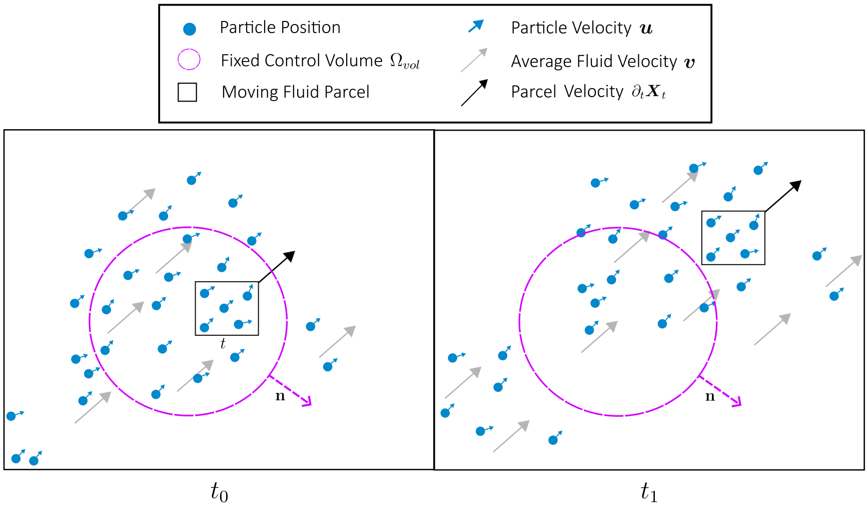

Particle Position
Fixed Control Volume Ω_{vol}
Moving Fluid Parcel
Particle Velocity $\boldsymbol{u}$
Average Fluid Velocity $\boldsymbol{v}$
Parcel Velocity $\partial_t \boldsymbol{X}_t$
t
n
n
t_0
t_1

FIGURE 2.1: Illustration of the different viewpoints of a moving fluid.

Note that this description not only considers global conservation, but also hints at the connection to directional movements of the quantity. In general, a conservation law provides a fundamental link between the changes of a quantity and its flux, and thus the velocity of the fluid.

Within this thesis, we focus on the conservation of mass (or density) without any sinks or sources present. The following sections are intended as an introduction to fluid densities and the continuity equation, a local and differential formulation of mass conservation. In doing so, we touch upon three different viewpoints:

1. A low level molecular point of view linking the distribution and movements of particles to fluid densities and velocities.

2. The Eulerian point of view considering changes of densities within control volumes fixed in space.

3. The Lagrangian point of view, considering the volume changes of moving parcels with fixed mass.

An illustration of these different viewpoints is provided in Figure 2.1. The introduction will be mostly set in a $d = 3$ dimensional physical space, but generalizations to higher dimensions are straightforward.

2.2.1 *Kinetic Theory: The Molecular Distribution*

The field of kinetic theory shows that the essential conservation laws of fluids can be derived from a microscopic (or molecular) viewpoint (Born & Green, 1946). That is, the dynamics of a fluid are described starting from a set of individual particles. Each particle moves randomly and thus defines a velocity, be it due to particle interactions, active particle movements, or a movement of the medium in which the particle resides. The basis of kinetic theory is the so-called *molecular distribution function* Ψ in phase space.

Definition 1. (Molecular Distribution Function)
Let $\boldsymbol{x} \in \Omega \subseteq \mathbb{R}^d$ with Ω be the spatial domain, and let $t \in [t_0, T] \subset \mathbb{R}_{\geq 0}$ be the time. The molecular distribution function $\Psi(t, \boldsymbol{x}, \boldsymbol{u})$ is defined such that

$$\int_{\Delta t} \int_{\Delta \boldsymbol{x}} \int_{\Delta \boldsymbol{u}} \Psi(t, \boldsymbol{x}, \boldsymbol{u}) \, \mathrm{d}\boldsymbol{u} \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t \tag{2.5}$$

is the probability that a molecule with a velocity $\boldsymbol{u} \in \mathbb{R}^3$ within $\Delta \boldsymbol{u} = \Delta u_1 \Delta u_2 \Delta u_3$ occupies the volume $\Delta \boldsymbol{x} = \Delta x_1 \Delta x_2 \Delta x_3$ in the time interval Δt. Consequently, after marginalizing out $\boldsymbol{u}$, we obtain with

$$\int_{\Delta t} \int_{\Delta \boldsymbol{x}} \Psi(t, \boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t \tag{2.6}$$

the probability that a particle with any velocity occupies the volume $\Delta \boldsymbol{x}$ in the timeframe Δt.

Based on the distribution function, it is possible to define common quantities such as the mass density ρ and the (local mean) velocity $\boldsymbol{v}$. Assuming that each particle has mass m, and there is a total of N particles, the density of the fluid is given by the function $\rho : [t_0, T] \times \Omega \mapsto \mathbb{R}_{\geq 0}$, defined by

$$\rho \equiv \int mN \, \Psi(t, \boldsymbol{x}, \boldsymbol{u}) \; \mathrm{d}\boldsymbol{u}, \qquad [\rho] = \left[\frac{\mathrm{kg}}{\mathrm{m}^3}\right]. \tag{2.7}$$

The average velocity, which we denote by $\boldsymbol{v}$, is defined indirectly via the flux $\rho \boldsymbol{v}$ given by the expected momentum

$$\rho \boldsymbol{v} \equiv \int \big(mN \, \Psi(t, \boldsymbol{x}, \boldsymbol{u})\big) \cdot \boldsymbol{u} \; \mathrm{d}\boldsymbol{u}, \qquad [\rho \boldsymbol{v}] = \left[\frac{\mathrm{kg}}{\mathrm{m}^2 \mathrm{s}}\right]. \tag{2.8}$$

The average velocity $\boldsymbol{v} : [t_0, T] \times \Omega \mapsto \mathbb{R}^3$ is simply referred to as the velocity of the fluid at a specified location and time. Note that the velocity is undefined for a density of zero.

In this molecular point of view, conservation of mass implies that no particles appear or disappear. That is, any change in mass within a fixed control volume must be explained by particles leaving or entering the volume. As the average movements of particles are described by their (coarse-grained) velocity, it is clear that mass conservation provides a fundamental link between density ρ and velocity $\boldsymbol{v}$.

2.2.2 *Continuity Equation for the Conservation of Mass*

With the relevant quantities defined, we continue with an introduction to the Eulerian specification of the continuity equation. We closely follow the introductions provided by Landau and Lifshitz (2013) and Hirsch (2007). Similar introductions can be found in any textbook that covers the fundamentals of computational fluid dynamics.

Let $\Omega_{vol} \subseteq \Omega$ be an arbitrary volume fixed in space (the *control volume*), bounded by a closed surface S (the *control surface*), with $\Omega \subset \mathbb{R}^3$ being the space domain. Furthermore, let $\boldsymbol{n} \in \mathbb{R}^3$ be a vector with unit length that is normal to the surface S. By convention, this vector $\boldsymbol{n}$ points outwards, as shown in Figure 2.1. The fluid mass flowing out of the control volume (per unit time) is given by the surface integral

$$\oint_S (\rho \boldsymbol{v}) \cdot \boldsymbol{n} \, \mathrm{d}S, \qquad \left[\frac{\mathrm{kg}}{\mathrm{s}}\right]. \tag{2.9}$$

A conservation law dictates that the change in a quantity for any control volume Ω_{vol} must be equal to all incoming fluxes plus possible sources. Assuming there are no sinks or sources of mass, we can write the conservation of mass in its integral form as

$$\underbrace{\frac{\partial}{\partial t} \int_{\Omega_{vol}} \rho \, \mathrm{d}\boldsymbol{x}}_{\text{change in total quantity}} = \underbrace{- \oint_S (\rho \boldsymbol{v}) \cdot \vec{n} \, \mathrm{d}S}_{\text{total incoming flux}} . \tag{2.10}$$

With the divergence theorem (also known as Gauss' theorem) the surface integral can be rewritten as volume integral:

$$\frac{\partial}{\partial t} \int_{\Omega_{vol}} \rho \, \mathrm{d}\boldsymbol{x} + \int_{\Omega_{vol}} \nabla \cdot (\rho \boldsymbol{v}) \, \mathrm{d}\boldsymbol{x} = 0 \tag{2.11}$$

$$= \int_{\Omega_{vol}} (\partial_t \rho + \nabla \cdot (\rho \boldsymbol{v})) \, \mathrm{d}\boldsymbol{x} = 0. \tag{2.12}$$

We follow the convention that the divergence and gradient are with respect to the spatial variables, i. e. $\nabla(\cdot) \equiv \nabla_x(\cdot)$. As this equation needs to hold for any arbitrary volume Ω_{vol} it has to be valid everywhere (Arbogast & Bona, 1999, Chapter 1 Proposition 1.39). That is, we can write the conservation law in its differential form, leading to the PDE referred to as continuity equation:

$$\partial_t \rho + \nabla \cdot (\rho \boldsymbol{v}) = 0. \tag{2.13}$$

The associated initial value problem (IVP) is commonly written in the following form, where an initial density ρ_{t_0} and the full velocity field $\boldsymbol{v}$ are known.

Definition 2. (Continuity Equation)
Let $\Omega \subset \mathbb{R}^d$ be an open set, $t_0 \in [0, T)$ with $T \in \mathbb{R}_+$, and let $\boldsymbol{v} : [t_0, T] \times \Omega \mapsto \mathbb{R}^d$ be a given velocity field. A density field $\rho : [t_0, T] \times \Omega \mapsto \mathbb{R}_{\geq 0}$ with initial density $\rho_{t_0} : \Omega \mapsto \mathbb{R}_{\geq 0}$ is said to fulfill the continuity equation with respect to $\boldsymbol{v}$ if it a solution to

$$\begin{cases} \partial_t \rho(t, \boldsymbol{x}) + \nabla \cdot (\boldsymbol{v}(t, \boldsymbol{x}) \rho(t, \boldsymbol{x})) = 0 & (t, \boldsymbol{x}) \in [t_0, T) \times \Omega, \\ \rho(t_0, \boldsymbol{x}) = \rho_{t_0}(\boldsymbol{x}) & \boldsymbol{x} \in \Omega. \end{cases} \tag{2.14}$$

We call a pair of $\boldsymbol{v}$ and $\hat{\rho}$ physically inconsistent if $\hat{\rho}$ is not a solution to Eq. (2.14).

Classical proofs for the existence and uniqueness of solutions to Eq. (2.14) commonly rely on basic regularity assumptions on the velocity such as boundedness and Lipschitz continuity, see e.g. Ambrosio and Crippa (2014).

Note that the continuity equation alone does not explain anything about the dynamics of a fluid. Rather, it provides a link between averaged particle velocities and density movements. The actual dynamics of the fluid are governed by the velocity field, which has to be fully specified for the IVP in Eq. (2.14). If only the initial velocity is known, additional equations that dictate the evolution of the velocity have to be provided.

2.2.3 *The continuity equation from a Lagrangian perspective*

The previous section introduced the continuity equation starting from a control volume that is fixed in space and time. This approach, termed the Eulerian perspective, observes the fluid flow from a fixed position. In contrast, the Lagrangian perspective tracks a fluid by moving along with it, similar to monitoring from a drifting buoy. More specifically, the fluid is described from the perspective of moving infinitesimal volumes with constant mass, the so-called fluid parcels. Fluid parcels should, however, not be confused with individual particles, as they still describe the fluid from a coarse-grained perspective.

In the following, we will express the continuity equation from the Lagrangian point of view, which describes how the density of a parcel changes alongside its trajectory.

Trajectory of a Parcel. First consider the movements of a Lagrangian parcel. Let $\boldsymbol{X}_t(\boldsymbol{x}) \in \Omega$ denote the position of the parcel at time t, with initial position $\boldsymbol{x}$. This initial position $\boldsymbol{x}$ can be seen as a continuous label to identify different parcels. We further know that, by definition, the parcel should move with the fluid, i. e. its change in position is given by the velocity $\boldsymbol{v}(\boldsymbol{X}_t(\boldsymbol{x}), t)$. The position of the parcel can then be described as the solution to an ordinary differential equation (ODE).

Definition 3. (Parcel Trajectory)
Let $\Omega \subset \mathbb{R}^d$ be an open set, $t_0 \in [0, T)$ with $T \in \mathbb{R}_+$, and let $\boldsymbol{v} : [t_0, T] \times \Omega \mapsto \mathbb{R}^d$ be a given velocity field. Then the function $\boldsymbol{X} : [t_0, T) \times \Omega \mapsto \Omega$ refers to the position of a parcel with initial position $\boldsymbol{x}$ and starting time t_0 if it is a solution to

$$\begin{cases} \partial_t \boldsymbol{X}_t\,(\boldsymbol{x}) = \boldsymbol{v}\,(\boldsymbol{X}_t(\boldsymbol{x}), t) & t \in [t_0, T), \\ \boldsymbol{X}_{t_0}(\boldsymbol{x}) = \boldsymbol{x} & \boldsymbol{x} \in \Omega. \end{cases} \tag{2.15}$$

We call the curve $t \mapsto \boldsymbol{X}(t)$ for a fixed $\boldsymbol{x}$ the trajectory of a parcel. We refer to $\boldsymbol{X} : [t_0, T] \times \Omega \mapsto \Omega$ as the flow (map) of $\boldsymbol{v}$ starting at time t_0.

Given $\boldsymbol{v}$ and a fixed $\boldsymbol{x}$, the flow exists and is unique following the Cauchy-Lipschitz Theorem (also known as the Picard-Lindelöf Theorem) under the assumption that $\boldsymbol{v}$ is bounded and uniformly Lipschitz in time (see Theorem 6 and Hartman (2002, Chapter 2)).

Density of a Parcel. Let the density of a parcel with initial position $\boldsymbol{x}$ at time t be given by $\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))$, i. e. it is now dependend on the position and path of a parcel. Using the chain rule and $\partial_t \boldsymbol{X}_t = \boldsymbol{v}$, the temporal change in parcel density is given by

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} = \partial_t \rho + (\nabla \rho) \cdot (\partial_t \boldsymbol{X}_t) = \partial_t \rho + (\nabla \rho) \cdot \boldsymbol{v}. \tag{2.16}$$

The continuity equation can be rewritten by expanding the divergence term using a generalized product rule:

$$\partial_t \rho + (\nabla \cdot (\rho \boldsymbol{v})) = \partial_t \rho + (\nabla \rho) \cdot \boldsymbol{v} + \rho(\nabla \cdot \boldsymbol{v}) = 0. \tag{2.17}$$

Inserting Eq. (2.16) into the expanded form of the continuity equation (Eq. (2.17)) then provides us the continuity equation in terms of the total derivative of ρ, i. e. from the perspective of a moving parcel:

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} + \rho(\nabla \cdot \boldsymbol{v}) = 0. \tag{2.18}$$

Log-Density of a Parcel. An alternative formulation in terms of the log-density of a parcel (assuming $\rho > 0$) can be obtained similarly by first applying the chain rule

$$\frac{\mathrm{d}\ln\rho}{\mathrm{d}t} = \frac{1}{\rho}\left(\partial_t\rho + (\nabla\rho)\cdot\boldsymbol{v}\right) \tag{2.19}$$

and then substituting Eq. (2.19) into Eq. (2.17):

$$\frac{\mathrm{d}\ln\rho}{\mathrm{d}t} + (\nabla\cdot\boldsymbol{v}) = 0 \qquad \Big[(\nabla\cdot\boldsymbol{v}) = \mathrm{tr}(J_{\boldsymbol{x}}(\boldsymbol{v}))\Big]. \tag{2.20}$$

In summary, the (log-)density of a parcel at time t with initial position $\boldsymbol{x}$ and initial density $\rho_0(\boldsymbol{x})$ is then given as the solution to an IVP.

Definition 4. (Continuity Equation: Lagrangian View)
Let $\Omega \subset \mathbb{R}^d$ be an open set, $t_0 \in [0,T)$ with $T \in \mathbb{R}_+$, and $\boldsymbol{v} : [t_0,T] \times \Omega \mapsto \mathbb{R}^d$ be a given velocity field. Let $\boldsymbol{X} : [t_0,T] \times \Omega \mapsto \Omega$ be the flow map of $\boldsymbol{v}$ starting at time t_0 according to Definition 3. A nonzero density field $\rho : [t_0,T] \times \Omega \mapsto \mathbb{R}_{>0}$ with initial density $\rho_{t_0} : \Omega \mapsto \mathbb{R}_{>0}$ is said to fulfill the Lagrangian form of the continuity equation if it is a solution to

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\ln\rho(t,\boldsymbol{X}_t(\boldsymbol{x})) = -\nabla\cdot\boldsymbol{v}(t,\boldsymbol{X}_t(\boldsymbol{x})) & (t,\boldsymbol{x}) \in [t_0,T)\times\Omega, \\ \ln\rho(t_0,\boldsymbol{X}_{t_0}(\boldsymbol{x})) = \ln\rho_{t_0}(\boldsymbol{x}) & \boldsymbol{x}\in\Omega. \end{cases} \tag{2.21}$$

In fact, solutions to Eq. (2.21) coincide with solutions to the continuity equation in Definition 2 for nonzero densities if the velocity field follows basic regularity assumptions such as boundedness and Lipschitz continuity (see Appendix Section B.1.4).

This formulation allows for an intuitive interpretation from the parcel perspective. The density of the parcel at any time is defined by (i.) its density at the initial time point and (ii.) the negative divergence of the velocity. The divergence affects the volume of the parcel, so the density can increase or decrease even though each parcel has a constant mass. Consequently, all we need for inferring the density of a parcel is its initial density, and

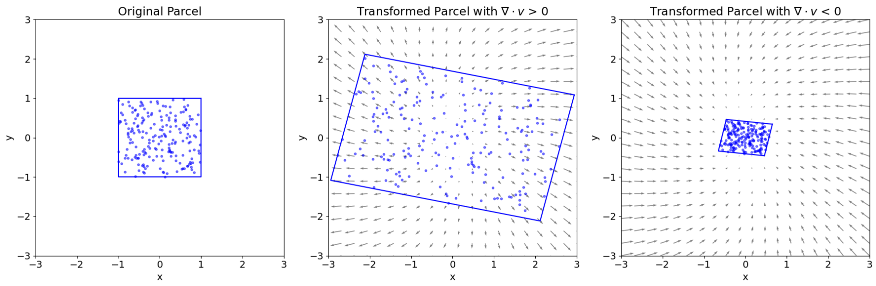


FIGURE 2.2: Transformation of a parcel with two different velocity fields. A positive divergence expands the volume, decreasing the density. A negative divergence contracts the volume, increasing the density.

how much the space was distorted due to the divergence of the velocity. For an illustration of the effect of vector fields with positive and negative divergence, see Figure 2.2.

2.2.4 *Special Cases of the Continuity Equation*

The continuity equation only describes a general transport of the density with a given velocity field. That is, additional equations defining the velocity are necessary to fully define the density movements. In a similar vein, a range of PDEs can be reformulated by deconstructing them into an evolving density with additional constraints on the velocity. We provide as examples the heat equation and a subset of the Fokker-Planck equations. Other dynamics that can be viewed in a similar manner include, for example, dynamical optimal transport (Benamou & Brenier, 2000), porous medium equations or collective dynamics (Carrillo et al., 2016; 2019).

Fokker-Planck Equation. The Fokker-Planck equation (FPE) describes the evolution of the probability of stochastic particles. To recall the FPE, first consider the stochastic trajectory of a Brownian particle subject to drift.

Definition 5. (Brownian motion with drift; (Särkkä & Solin, 2019, Sect. 5.2)) *Let $\boldsymbol{x}(t) \in \mathbb{R}^d$ be a stochastic process with initial distribution p_0 be defined as the solution to the stochastic differential equation (SDE)*

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{\mu}(t, \boldsymbol{x})\,\mathrm{d}t + D(t, \boldsymbol{x})\,\mathrm{d}W_t \tag{2.22}$$

$$\boldsymbol{x}(t_0) \sim p_0(\boldsymbol{x}(t_0)) \tag{2.23}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^d$ *is the state,* $\boldsymbol{\mu}(t, \boldsymbol{x})$ *is a vector-valued function called drift, and* W_t *is a Brownian motion with diffusion matrix* Q *such that*

$$W_{t_{k+1}} - W_{t_k} \sim \mathcal{N}(\mathbf{0}, Q\Delta t_k).$$

Theorem 1. (FPE; adapted from Särkkä and Solin (2019, Theorem 5.4))
The probability density $p(t, \boldsymbol{x})$ *of the solutions of the SDE in Definition 5 solves the Fokker-Plank equation. Specifically, it is a solution to the IVP*

$$\begin{cases} \partial_t p(t, \boldsymbol{x}) = -\nabla \cdot [\boldsymbol{\mu}(t, \boldsymbol{x}) p(t, \boldsymbol{x})] \\ \qquad\qquad + \dfrac{1}{2} \displaystyle\sum_{i,j} \frac{\partial^2}{\partial x_i \partial x_j} \left\{ \left[D(t, \boldsymbol{x}) Q D(t, \boldsymbol{x})^\top \right]_{ij} p(t, \boldsymbol{x}) \right\} \\ \\ p(t_0, \boldsymbol{x}) = p_0(\boldsymbol{x}) \end{cases}$$

Proof. See Särkkä and Solin (2019) Section 5.2. □

Theorem 2. (FPE with time-dependent diagonal diffusion)
Let $p(t, \boldsymbol{x})$ *be a solution to the Fokker-Planck equation 1 with a matrix valued* $D(t, \boldsymbol{x}) Q D(t, \boldsymbol{x})^\top$ *that is diagonal and depends only on time* t*. That is, let* $\boldsymbol{s} : t \mapsto \mathbb{R}^d$ *denote a vector-valued function such that* $\operatorname{diag}(\boldsymbol{s}(t)) = D(t, \boldsymbol{x}) Q D(t, \boldsymbol{x})^\top$*. Then, the changes in density* $\partial_t p(t, \boldsymbol{x})$ *correspond to the changes given by the continuity equation in Definition 2 with velocity field*

$$\boldsymbol{v}(t, \boldsymbol{x}) = \boldsymbol{\mu}(t, \boldsymbol{x}) - \frac{\boldsymbol{s}(t)}{2} \nabla \Big(\ln \rho(t, \boldsymbol{x}) \Big). \tag{2.24}$$

Proof.

$$\begin{aligned} \partial_t \rho(t, \boldsymbol{x}) &= -\nabla \cdot (\rho(t, \boldsymbol{x}) \boldsymbol{v}(t, \boldsymbol{x})) \\ &= -\nabla \cdot \Big(\rho(t, \boldsymbol{x}) \big[\boldsymbol{\mu}(t, \boldsymbol{x}) - \frac{\boldsymbol{s}(t)}{2} \nabla (\ln \rho(t, \boldsymbol{x})) \big] \Big) \\ &= -\nabla \cdot \Big(\rho(t, \boldsymbol{x}) \boldsymbol{\mu}(t, \boldsymbol{x}) \Big) + \frac{1}{2} \nabla \cdot \Big(\boldsymbol{s}(t) \rho \frac{1}{\rho} \nabla \rho \Big) \\ &= -\nabla \cdot \Big(\rho(t, \boldsymbol{x}) \boldsymbol{\mu}(t, \boldsymbol{x}) \Big) + \frac{\boldsymbol{s}(t)}{2} \Delta \rho(t, \boldsymbol{x}) \\ &= -\nabla \cdot \Big(\rho(t, \boldsymbol{x}) \boldsymbol{\mu}(t, \boldsymbol{x}) \Big) + \frac{1}{2} \sum_i \frac{\partial^2}{\partial x_i^2} \Big\{ [\operatorname{diag}(\boldsymbol{s}(t))]_{ii} \, \rho(t, \boldsymbol{x}) \Big\} \end{aligned}$$

□

The Heat Equation. As another example, consider the heat equation.

Definition 6. (Heat Equation)
Let $\Omega \subset \mathbb{R}^d$ be an open set and $t_0 \in [0, T)$ with $T \in \mathbb{R}_+$. A nonzero density field $\rho : [t_0, T] \times \Omega \mapsto \mathbb{R}_{>0}$ with initial density $\rho_{t_0} : \Omega \mapsto \mathbb{R}_{>0}$ is said to fulfill the heat equation if it is a solution to

$$\begin{cases} \partial_t \rho(t, \boldsymbol{x}) = \kappa \Delta \rho(t, \boldsymbol{x}) & (t, \boldsymbol{x}) \in [t_0, T) \times \Omega \\ \rho(t, \boldsymbol{x}) = \rho_{t_0}(\boldsymbol{x}) & \boldsymbol{x} \in \Omega \end{cases} \tag{2.25}$$

The dynamics of the heat equation can be seen as a special case of the Fokker-Planck equation discussed in Theorem 2, with $\boldsymbol{\mu}(t, \boldsymbol{x}) \equiv \mathbf{0}$ and $s(t) = 2\kappa$. The constant $\kappa \in \mathbb{R}_+$ is commonly referred to as the diffusion coefficient. Consequently, it can also be written as a continuity equation with the velocity given by

$$v(t, \boldsymbol{x}) = -\kappa \nabla (\ln \rho). \tag{2.26}$$

2.3 NEURAL NETWORKS

Deep neural networks have arisen as one of the most prominent methods and widely used methods for modeling in Machine Learning. Reasons for the popularity can be found in their ability to train on large amounts of data through efficient use of modern hardware (Krizhevsky et al., 2012; Raina et al., 2009), the rich software ecosystem that allows fast prototyping (Abadi et al., 2016a; Jia et al., 2014; Paszke et al., 2019), the ability to automatically learn high-level features (LeCun et al., 2015), and finally their generalization capabilities in heavily overparameterized settings (Nakkiran et al., 2021).

On a fundamental level, a neural network $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ is a composition of parameterized functions.

$$\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{f}_k \circ \boldsymbol{f}_{k-1} \circ \cdots \circ \boldsymbol{f}_2 \circ \boldsymbol{f}_1(\boldsymbol{x}), \tag{2.27}$$

where $k - 1$ is the total number of hidden layers, $\boldsymbol{f}_j(\boldsymbol{x}; \boldsymbol{\theta})$ the j-th *layer* in the composition, and $\boldsymbol{x}$ a single datum as input. The parameters $\boldsymbol{\theta}$ depend on the specific structure of the layers. For the sake of notational brevity, we sometimes omit the explicit dependence on $\boldsymbol{\theta}$, i. e., $\boldsymbol{f}_j(\boldsymbol{x}; \boldsymbol{\theta}) := \boldsymbol{f}_j(\boldsymbol{x})$.

A standard feedforward neural network is a special case of Eq. (2.27), where each $\boldsymbol{f}_j$ consists of an affine transformation parameterized by $\boldsymbol{W}_j \in$

$\mathbb{R}^{d_j \times d_{j-1}}$ and $\boldsymbol{b}_j \in \mathbb{R}^{d_j}$ followed by (nonlinear) element-wise activation functions σ_j:

$$\boldsymbol{f}_j(\boldsymbol{x};\boldsymbol{\theta}) = \sigma_j(\boldsymbol{W}_j(\boldsymbol{x}) + \boldsymbol{b}_j), \tag{2.28}$$

with the learnable *parameters* referring to $\boldsymbol{\theta} = \{(\boldsymbol{W}_j, \boldsymbol{b}_j)\}_{j=1}^{k}$. Parameters are initialized randomly before training (see e.g. He et al. (2015)). Activations are commonly chosen to be fixed non-linearities, such as the ReLU ($\sigma(x) := \max\{0, x\}$) or SiLU/Swish ($\sigma(x) := x \cdot \frac{1}{1+\exp^{-x}}$). The last activation σ_k is selected based on the domain of the target variable(s) $\boldsymbol{y}$, and can, for example, be the identity for unbounded regression or the exponential function to enforce positive outputs.

The training of neural networks (i. e. optimizing $\boldsymbol{\theta}$) revolves around the minimization of a differentiable loss function $L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{Y}) := L(\boldsymbol{\theta})$, which quantifies the difference between the network predictions and the actual target values. Aside from a data loss, additional terms might be added to weakly enforce additional constraints or regularization. Neural networks are typically optimized through gradient-based optimization algorithms like stochastic gradient descent (SGD) and its variants. A simplified update scheme for gradient descent can be written as follows, where $\alpha \in \mathbb{R}_{>0}$ denotes a small constant commonly referred to as learning rate:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{2.29}$$

Consequently, efficiently calculating the gradient of function compositions is an integral part of deep learning. This can be accomplished by the application of the chain rule combined with bookkeeping of the forward-pass results. The method is more generally known as backpropagation in context of deep learning and we refer to introductory Machine Learning textbooks such as Goodfellow et al. (2016) for a detailed treatment of this topic.

2.3.1 *Residual Networks*

Building upon the foundation of fully connected neural networks, He et al. (2016a) and He et al. (2016b) introduced residual networks (ResNets). The underlying motivation was to build deeper neural network architectures without suffering from the degradations that occurred for very deep network architectures. In a residual network, each additional layer has to learn

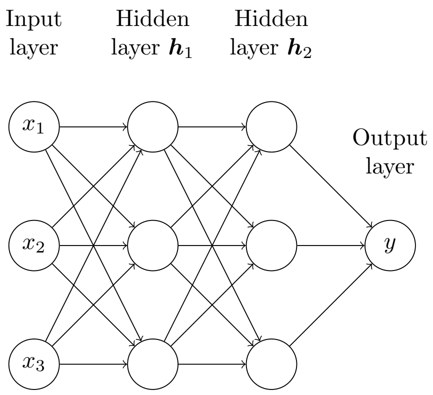


FIGURE 2.3: Illustration of a neural networks with 3 input neurons, 2 hidden layers, and a single output neuron.

only a correction term of the previous prediction. Following the notation of Eq. (2.27), each residual layer can be written as

$$f_j(\boldsymbol{x}) = \boldsymbol{x} + g_j(\boldsymbol{x}), \tag{2.30}$$

where each g_j is a sequence of dense layers that keeps the dimensionality, i. e. $\dim(\boldsymbol{x}) = \dim(\boldsymbol{g}(\boldsymbol{x}))$. The hidden states $\boldsymbol{h}_j$ can then be written as an update equation:

$$\boldsymbol{h}_{j+1} = \boldsymbol{h}_j + g_j(\boldsymbol{h}_j), \qquad \boldsymbol{h}_0 = \boldsymbol{x}. \tag{2.31}$$

In this notation, an interesting connection to the solution of dynamical systems becomes apparent. Instead of discrete layers, consider $g_j(\boldsymbol{x})$ as a function continuous in j, i. e. $g_j(\boldsymbol{x}) : [0, k] \times \mathbb{R}^d \to \mathbb{R}^d$ and let, with some abuse of notation,

$$g_j(\boldsymbol{x}) := \Delta j \cdot g_j(\boldsymbol{x}),$$

with $\Delta j = 1$. In addition, let $\boldsymbol{h}_j := \boldsymbol{h}_j(\boldsymbol{x})$, as the hidden value $\boldsymbol{h}$ depends on both j and the initial input $\boldsymbol{x}$. Then each residual layer from Eq. (2.30) resembles an explicit Euler step for solving the ordinary differential equation

$$\partial_j \boldsymbol{h}_j(\boldsymbol{x}) = \boldsymbol{g}(j, \boldsymbol{h}_j(\boldsymbol{x})), \qquad \boldsymbol{h}_0(\boldsymbol{x}) = \boldsymbol{x}, \tag{2.32}$$

with the input $\boldsymbol{x}$ serving as initial condition. That is, each layer represents the dynamics at a discrete time step j. This perspective of modern neural networks as a numerical discretization of a differential equation was first established by Weinan (2017) and subsequent work took advantage of this view to propose new network designs (Haber & Ruthotto, 2017; Y. Lu et al., 2018; Ruthotto & Haber, 2020).

2.3.2 *Infinitely Deep Neural Networks: Neural-ODEs*

Motivated by the link between residual networks and ODEs, R. T. Q. Chen et al. (2018) propose to explicitly learn continuous dynamical systems for modeling data, resulting in *neural ODEs*. Instead of only interpreting a neural network as a discretized solution to a dynamical system, a neural ODE is directly defined as $\boldsymbol{f}(\boldsymbol{x};\theta) = \boldsymbol{h}_T(\boldsymbol{x};\theta)$ with $\boldsymbol{h}_T$ being the solution to the initial value problem

$$\begin{cases} \partial_t \boldsymbol{h}_t(\boldsymbol{x};\theta) = \boldsymbol{g}(t, \boldsymbol{h}_t(\boldsymbol{x};\theta);\theta) & t \in [t_0, T), \\ \boldsymbol{h}_{t_0}(\boldsymbol{x};\theta) = \boldsymbol{x} & \boldsymbol{x} \in \mathbb{R}^d, \end{cases} \tag{2.33}$$

with $0 < t_0 < T$, where $\boldsymbol{g} : [t_0, T) \times \mathbb{R}^d \mapsto \mathbb{R}^d$ is a trainable neural network that takes as input continuous time t and a hidden state. The hidden state $\boldsymbol{h}_t : [t_0, T] \times \mathbb{R}^d \mapsto \mathbb{R}^d$ is a function that evolves over time t and depends on the initial input $\boldsymbol{x}$.

Note that this resembles the trajectory of a Lagrangian parcel from Definition 3, although in a higher dimensional space. Similarly to the parcel trajectory, the existence and uniqueness of Eq. (2.33) require Lipschitz continuous and bounded dynamics $\boldsymbol{g}$. This is in practice easily achieved with common neural network architectures.

The solution of Eq. (2.33) can be obtained with any numerical ODE solver. R. T. Q. Chen et al. (2018) specifically suggest adaptive Runge-Kutta methods, as they can automatically adapt their precision depending on the dynamics given by the network.

For differentiating through the ODE solution, methods can be mainly divided into two categories: The *discretize then optimize* (DTO), and the *optimize then discretize* (OTD) approach. DTO relies on first discretizing the solution with a numerical solver and then backpropagating through it. However, this approach may result in a large computational graph and high memory requirements due to the many individual steps of the solver. To avoid this, R. T. Q. Chen et al. (2018) propose an efficient autograd implementation of the adjoint sensitivity method (Cacuci, 1981a; 1981b; Pontryagin, 1987) to calculate the gradients $\nabla_\theta \ell(\boldsymbol{f}(\boldsymbol{x};\theta), \boldsymbol{y})$, which is an OTD approach. A comprehensive discussion and excellent overview of neural differential equations and the calculation of their gradients can be found in Kidger (2021).

2.4 NORMALIZING FLOWS

Remark. *The Lagrangian flow networks presented in Chapter 4 rely heavily on concepts related to Normalizing Flows (NFs). In this section, we provide a general overview of (conditional) NFs, and go into some detail for the invertible residual networks that are used in Chapter 4. In addition, continuous Normalizing Flows are relevant in context of the semi-Lagrangian data assimilation, which serves as a baseline comparison to Lagrangian flow networks. We note that parts of this section are based on the related work of our publication Torres et al. (2024).*

Normalizing Flows (NFs) are a general approach to warping a simple probability distribution into a more complex target distribution with invertible and differentiable transformations, i. e., diffeomorphisms. Let $\mathbf{Z} \in \mathbb{R}^d$ be a random variable with a known density function $\mathbf{Z} \sim p_{\mathbf{Z}}(z)$ and let $\mathbf{X} = \mathcal{T}^{-1}(\mathbf{Z}; \theta)$ where $\mathcal{T}$ is a diffeomorphism with trainable parameters. We will often omit the explicit dependence of $\mathcal{T}$ on the parameters for the sake of notational brevity. With a change of variables the probability density of $\mathbf{X}$ can be expressed in terms of the *base density* $p_{\mathbf{Z}}$, the map $\mathcal{T}$, and the determinant of its Jacobian:

$$p_{\mathbf{X}}(\boldsymbol{x}; \theta) = p_{\mathbf{Z}}(\mathcal{T}(\boldsymbol{x})) \left| \det J\mathcal{T}(\boldsymbol{x}) \right|. \tag{2.34}$$

A key property of bijections is that the Jacobian determinant can be factorized into their individual Jacobian determinants. That is, the determinant of the Jacobian of $\mathcal{T} = \mathcal{T}_1 \circ \cdots \circ \mathcal{T}_{k-1} \circ \mathcal{T}_k$ is given by

$$\left| \det J \left(\mathcal{T}_1 \circ \cdots \circ \mathcal{T}_{k-1} \circ \mathcal{T}_k \right)(\boldsymbol{x}) \right| = \left| \prod_{j=1}^{k} \det J\mathcal{T}_j(\boldsymbol{h}_{j-1}) \right| \tag{2.35}$$

where $\boldsymbol{h}_j = \mathcal{T}_j \circ \mathcal{T}_{j-1} \circ \cdots \circ \mathcal{T}_1(\boldsymbol{x})$ are the intermediate maps of $\boldsymbol{x}$ and $\boldsymbol{h}_0 = \boldsymbol{x}$. The Jacobian determinant of $\mathcal{T}$ can then be computed cheaply as long as the layer-wise Jacobian determinants are cheap. Algorithm 1 provides a description for evaluating the (log-) density at a single data point, and Algorithm 2 describes the sampling process (optionally also providing the density of the sample). A visualization of the transformations in both directions is shown in Fig. 2.4. To avoid the expensive computation of the Jacobian determinant with autograd, NF layers typically rely on

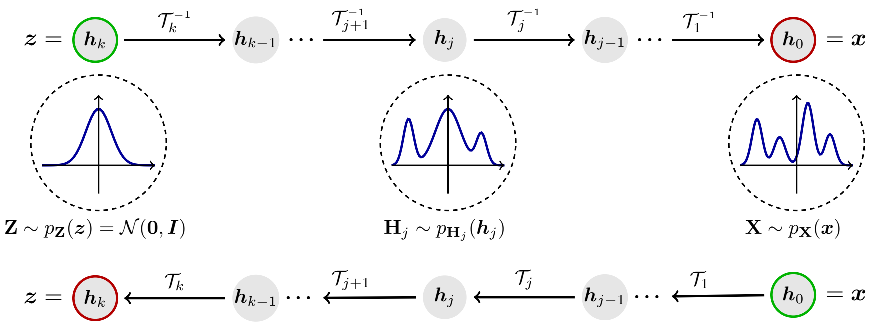


FIGURE 2.4: A Normalizing Flow warps a known base distribution $p_{\mathbf{Z}}$ into a more complex distribution $p_{\mathbf{Z}}$ via diffeomorphisms $\mathcal{T}$.

transformations with Jacobian determinants that are cheap to evaluate (and differentiate). The Jacobian determinant of the inverse can be computed by the Jacobian determinant of forward transformation with the identity

$$\det\left(J\mathcal{T}^{-1}(z)\right) = \det\left(\left[J\mathcal{T}(x)\right]^{-1}\right) = \frac{1}{\det\left(J\mathcal{T}(x)\right)}, \tag{2.36}$$

with $x = \mathcal{T}^{-1}(z)$.

For a comprehensive review of Normalizing Flows, their applications, and various extensions, we refer to Kobyzev et al. (2020) and Papamakarios et al. (2021).

Direction of the Flow. As Normalizing Flows are based on invertible transforms, it does (formally) not matter which direction is the forward ($\mathcal{T}$) or inverse ($\mathcal{T}^{-1}$) transformation. In the NF literature, the generative direction $\mathbf{Z} \to \mathbf{X}$ is commonly referred to as the forward direction of the flow, regardless of the actual implementation. In practice, inverting a layer often adds additional computational costs if no analytical inverse is available. Consequently, one direction of the transformation might be preferable. To add consistency between notation and implementation, we thus denote the direction that should be more efficient as the forward transformation $\mathcal{T}$. In our setting, the interest lies in evaluating the density $p_{\mathbf{X}}$ efficiently. Thus, we define that the forward transfom (or forward flow) $\mathcal{T}$ corresponds to the direction $\mathbf{X} \to \mathbf{Z}$.

Algorithm 1 Evaluating the log-density of a Normalizing Flow.

```
Input: $x_p$; $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k$; $p_{\mathbf{Z}}(z)$;
Output: Log-Density $\ln p_{\mathbf{X}}(x_p; \theta)$

1: $\boldsymbol{h}_0 \leftarrow \boldsymbol{x}_p$
2: logabsdet $\leftarrow 0$
3: for $j \leftarrow 1$ to $k$ do
4:     $\boldsymbol{h}_j \leftarrow \mathcal{T}_j(\boldsymbol{h}_{j-1})$
5:     logabsdet $\leftarrow$ logabsdet $+ \ln \left| \det J\mathcal{T}_j(\boldsymbol{h}_{j-1}) \right|$
6: end for
7: $\boldsymbol{z} \leftarrow \boldsymbol{h}_k$
8: logdensity $\leftarrow \ln p_{\mathbf{Z}}(z) +$ logabsdet
9: return logdensity
```

Algorithm 2 Sampling from a Normalizing Flow.

```
Input: $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k$; $p_{\mathbf{Z}}(z)$;
Output: Sample $x_s$ of $\mathbf{X} \sim p_{\mathbf{X}}(x; \theta)$; Log-Density $\ln p_{\mathbf{X}}(x_s; \theta)$

1: $z_s \leftarrow$ sample from $p_{\mathbf{Z}}(z)$
2: $\boldsymbol{h}_k \leftarrow z_s$
3: logabsdet $\leftarrow 0$
4: for $j \leftarrow k$ to $1$ do
5:     $\boldsymbol{h}_{j-1} \leftarrow \mathcal{T}_j^{-1}(\boldsymbol{h}_j)$
6:     logabsdet $\leftarrow$ logabsdet $+ \ln \left| \det J\mathcal{T}_j^{-1}(\boldsymbol{h}_j) \right|$
7: end for
8: $\boldsymbol{x}_s \leftarrow \boldsymbol{h}_0$
9: logdensity $\leftarrow \ln p_{\mathbf{Z}}(z_s) +$ logabsdet
10: return $x_s$, logdensity
```

Jacobian Structure. Normalizing Flow layers are sometimes classified in terms of the structure of the Jacobian matrix (Kobyzev et al., 2020; Papamakarios et al., 2021), as this structure can simplify the computation of the determinant. For example, if the Jacobian matrix is diagonal (elementwise transformations) or triangular (autoregressive transformations), the determinant is given by the product of the diagonal Jacobian entries. If instead no restrictions are put onto the Jacobian structure, this is commonly referred to as a free-form Jacobian.

Architectures for Normalizing Flows. As we show in the following section, it is possible to build invertible neural networks for Normalizing Flows that resemble multilayer perceptrons. That is, one can use (invertible) affine transformations followed by (invertible) elementwise non-linearities. However, unlike general neural network architectures, the affine transformations for bijective layers retain the dimension of the input by necessity. As a consequence, they are limited in their flexibility, especially for low-dimensional inputs. Even if paired with common non-linearities such as the tanh, one would require many such layers in practice. Invertible residual networks (Behrmann et al., 2019) provide one way to overcome this limitation, as they allow intermediate mappings into high-dimensional spaces. Another approach to bypassing the limitation is to rely on more flexible and learnable nonlinearities (Durkan et al., 2019; Negri et al., 2023).

2.4.1 *Bijective Layers for Normalizing Flows*

This subsection provides a brief overview of the bijective transforms used in Chapter 3, covering basic affine transforms and transforms based on invertible residual networks.

2.4.1.1 *Affine Layers*

The simplest bijective layers are based on affine transformations

$$\begin{aligned} \mathcal{T}_{\text{AFF}}(\boldsymbol{x}) &= W\boldsymbol{x} + \boldsymbol{b}, \\ \mathcal{T}_{\text{AFF}}^{-1}(\boldsymbol{z}) &= W^{-1}(\boldsymbol{z} - \boldsymbol{b}), \\ |\det\left(J\mathcal{T}_{\text{AFF}}(\boldsymbol{x})\right)| &= |\det W|, \end{aligned} \tag{2.37}$$

where $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{b} \in \mathbb{R}^d$ and $W \in \mathbb{R}^{d\times d}$ is an invertible matrix with $\det(W) \neq 0$. To parameterize such invertible matrices, one may rely on common matrix decompositions, such as the QR, SVD, or LU decomposition. In the following we provide an overview of some special instances of affine bijective layers.

Layer Normalization. As an alternative to batch normalization, Kingma and Dhariwal (2018) propose the use of the so-called activation normalization (*actnorm*). The motivation is a more stable normalization layer for small mini-batch sizes. It is based on an affine transformation with a simple shift and scale given by

$$\begin{aligned} \mathcal{T}_{\mathrm{LN}}(\boldsymbol{x}) &= \operatorname{diag}(\boldsymbol{s})\ \boldsymbol{x} + \boldsymbol{b}, \\ \mathcal{T}_{\mathrm{LN}}^{-1}(\boldsymbol{z}) &= \operatorname{diag}(1/\boldsymbol{s})\ (\boldsymbol{z} - \boldsymbol{b}), \\ |\det(J\mathcal{T}_{\mathrm{LN}}(\boldsymbol{x}))| &= \prod_{i=1}^{d} s_i, \end{aligned} \tag{2.38}$$

with $\boldsymbol{s} \in \mathbb{R}^d_{>0}$ and $\boldsymbol{b} \in \mathbb{R}^d$.

The shift $\boldsymbol{b}$ and the scale $\boldsymbol{s}$ are initialized with the mean and standard deviation of the first minibatch passed to the layer. After this initialization, both vectors are left as freely learnable parameters.

Orthogonal Layers. Linear transforms with orthogonal matrices can be parameterized indirectly via Householder reflections. To our knowledge, Tomczak and Welling (2016) were the first to make use of Householder transforms for Normalizing Flows, with the objective of improving variational autoencoders. Specifically, any orthogonal $d \times d$ matrices can be decomposed into a product of d Householder matrices (Uhlig, 2001), where a Householder matrix is defined as

$$H = I_d - 2\boldsymbol{v}\boldsymbol{v}^\top, \tag{2.39}$$

with $\boldsymbol{v} \in \mathbb{R}^d$ and $\|\boldsymbol{v}\| = 1$. An invertible Householder transformation can then be constructed with

$$\mathcal{T}_{\mathrm{H}}(\boldsymbol{x}) = H\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{v}(\boldsymbol{v}^\top \boldsymbol{x}) \tag{2.40}$$

$$\mathcal{T}_{\mathrm{H}}^{-1}(\boldsymbol{z}) = H^{-1}\boldsymbol{z}. \tag{2.41}$$

The determinant of a Householder matrix is -1 (see e.g Matrix Cookbook Eq. 24 (Petersen, Pedersen, et al., 2008)), and thus

$$|\det(J\mathcal{T}_{\mathrm{H}}(\boldsymbol{x}))| = 1. \tag{2.42}$$

Householder transforms further enable a range of more flexible bijective layers, such as Sylvester transforms (Berg et al., 2018) or linear transformations based on QR (Hoogeboom et al., 2019) and SVD (Mathiasen et al., 2020) decompositions.

SVD Layers. With orthogonal matrices available, general parameterizations of matrices based on the singular value decomposition (SVD) can be used. That is, any (in this case square) matrix W can be decomposed with an SVD as

$$W = USV^\top, \tag{2.43}$$

where $U \in \mathbb{R}^{d\times d}$ and $V \in \mathbb{R}^{d\times d}$ are orthogonal matrices, and $S = \operatorname{diag}(\boldsymbol{s})$ is a positive diagonal matrix (Strang, 2022) with diagonal entries $\boldsymbol{s} \in \mathbb{R}^d_{\geq 0}$. Note that we restrict ourselves to square matrices $W \in \mathbb{R}^{d\times d}$ and positive singular values $\boldsymbol{s} \in \mathbb{R}^d_{>0}$ since the resulting matrix should be invertible. The orthogonal transforms, i.e. $V^\top \boldsymbol{x}$ and $U\boldsymbol{z}$ with $\boldsymbol{z} = SV^\top \boldsymbol{x}$, are each implemented with d consecutive Householder projections. The entries in $\boldsymbol{s} \in \mathbb{R}^d_{>0}$ provide the singular values of A. This leads to a bijective SVD transformation given by

$$\mathcal{T}_{\mathrm{SVD}}(\boldsymbol{x}) = W\boldsymbol{x} = USV^\top \boldsymbol{x} \tag{2.44}$$

$$\mathcal{T}^{-1}_{\mathrm{SVD}}(\boldsymbol{z}) = VS^{-1}U^\top \boldsymbol{z}. \tag{2.45}$$

The determinant is given by

$$|\det(J\mathcal{T}_{\mathrm{SVD}}(\boldsymbol{x})| = |\det(S)| = \prod_{i=1}^{d} [S]_{ii}. \tag{2.46}$$

A conceptually similar parameterization for linear layers is explored in Zhang et al. (2018). Their focus lies, however, on controlling the Lipschitz constant of general, non-invertible, networks for stabilizing their gradients.

In our implementations used in Chapter 4 we enforce positive values of $\boldsymbol{s}$ with an exponential transform, and do not restrict the Lipschitz constant of the SVD layers.

2.4.1.2 *Invertible Residual Networks*

In Section 2.3.1, the similarity of residual neural networks to Euler discretizations of a differential equation was discussed. Recall that solution maps to differential equations such as Eq. (2.32) are bijective under mild regularity assumptions for the dynamics, which is a consequence of the classical Cauchy-Lipschitz (or Picard-Lindelöf) Theorem. Consequently, it is reasonable to consider residual layers of the following form for invertible architectures

$$\mathcal{T}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{g}(\boldsymbol{x}), \tag{2.47}$$

with $\boldsymbol{g} : \mathbb{R}^d \mapsto \mathbb{R}^d$ given by a neural network. Similar to before, the complete network would resemble a forward Euler discretization. However, while the exact solution to the ODE in Eq. (2.32) is invertible, the discretized solution given by the residual network is not necessarily invertible.

One way to ensure invertibility of Eq. (2.47) is proposed by Behrmann et al. (2019), with the resulting model being called invertible residual networks (i-ResNet). To do so, they leverage that $\mathrm{Lip}(\boldsymbol{g}_j) < 1$ is a sufficient condition for invertibility due to the Banach fixed-point theorem. It is then guaranteed that a fixed-point iteration at each layer converges to the inverse of the residual layer in Eq. (2.47).

Invertible residual network layers. Given a neural network $\boldsymbol{g} : \mathbb{R}^d \mapsto \mathbb{R}^d$ with $\mathrm{Lip}(\boldsymbol{g}) < 1$, an invertible residual network layer is constructed as

$$\mathcal{T}_{\mathrm{RES}}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{g}(\boldsymbol{x}) \tag{2.48}$$

$$\mathcal{T}^{-1}_{\mathrm{RES}}(\boldsymbol{z}) = \lim_{m \to \infty} \boldsymbol{h}^{(m)}. \tag{2.49}$$

where the inverse is obtained as the fix point solution to

$$\boldsymbol{h}^{(0)} = \boldsymbol{z}, \qquad \boldsymbol{h}^{(m+1)} = \boldsymbol{z} - \boldsymbol{g}(\boldsymbol{h}^{(m)}). \tag{2.50}$$

Enforcing the Lipschitz constant. The challenge is then to enforce the Lipschitz constant of a neural network in practice. Behrmann et al. (2019) rely on the fact that the Lipschitz constant of a function composition is upper-bounded by the product of the individual functions' Lipschitz constants. Thus, they (i) use activation functions with a Lipschitz constant of one (e.g. tanh) and (ii) apply spectral normalization (Miyato et al., 2018) to ensure that the affine transformations' Lipschitz constant remains below one. That is, given an estimate $\hat{s}_j \approx \|\boldsymbol{W}_j\|$ obtained by the power iteration method, each layer of $\boldsymbol{g}$ is normalized via

$$\boldsymbol{W}_j := \begin{cases} c\boldsymbol{W}_j / \hat{s}_j & \text{if } c/\hat{s}_j < 1, \\ \boldsymbol{W}_j & \text{else,} \end{cases} \tag{2.51}$$

with $c < 1$ as a scaling parameter, resulting in $\|\boldsymbol{W}_j\| < 1$ and thus $\mathrm{Lip}(\boldsymbol{g}) < 1$. It should be noted that $\hat{s}_j$ is an underestimate and thus does not guarantee the bound on the Lipschitz constant. However, in practice, it was shown that the estimate is still sufficient to enforce the constraint reliably. In the implementation used for the LFlows in Chapter 4, as well as in our python package “Flow Conductor”, we enforce the Lipschitz constant each time the weights are accessed. The code is based on an adjusted version of the PyTorch spectral normalization hook `torch.nn.utils.parametrizations.spectral_norm` [1], which is based on Miyato et al. (2018).

Estimating the Jacobian Determinant. A difficulty in i-ResNets is the efficient computation of the Jacobian determinant in high dimension. For the low-dimensional settings considered in this thesis (≤ 3 dimensions), it is still feasible to compute the Jacobian and its determinant by brute force (i. e. with automatic differentiation). For experiments concerning LFlows, we generally relied on a brute force computation of the Jacobian determinant for all layers relying on invertible residual networks, unless explicitely stated otherwise.

For the sake of completeness, we shortly discuss the estimation of the Jacobian determinant of invertible residual networks which would become necessary in higher-dimensional settings. While there is no efficient closed-form of the Jacobian determinant for i-ResNets, it was shown that the log absolute Jacobian determinant corresponds to the Power series

$$\ln \left| \det(J\mathcal{T}_{\mathrm{RES}}(x)) \right| = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{\operatorname{tr}([J\mathcal{T}(x)]^i)}{i}. \tag{2.52}$$

Behrmann et al. (2019) propose to efficiently estimate the trace of the matrix power using the stochastic Hutchinson trace estimator (Hutchinson, 1989). For performance reasons, the infinite sum is usually cut off after 5-10 iterations, resulting in a biased estimate of the Jacobian determinant. Subsequent work solved this problem by randomizing the infinite sum, leading to an unbiased estimator (R. T. Chen et al., 2019).

Invertible DenseNet. For improving the model performance of i-ResNets, Perugachi-Diaz et al. (2021) propose invertible DenseNets (i-DenseNets). These provide an alternative architecture of the (previously fully connected) Lipschitz-constrained networks g. Specifically, i-DenseNets increase the hidden dimension within the residual blocks via concatenations, and accord-

1 https://pytorch.org/docs/stable/generated/torch.nn.utils.parametrizations.spectral_norm.html; Accessed 26.03.2024.

ingly adjust the activations such that the required Lipschitz constraints are still fulfilled. Consider the following high-level structure of an i-DenseNet residual layer:

$$g(\boldsymbol{x}) = W_{m+1} \circ \boldsymbol{\phi}_n \circ \cdots \circ \boldsymbol{\phi}_1(\boldsymbol{x}), \tag{2.53}$$

with $\boldsymbol{\phi}$ denoting a *DenseBlock*. Each *DenseBlock* is a concatenation of the input with a non-linear transformation of the input:

$$\boldsymbol{\phi}_1(x) = \begin{bmatrix} \boldsymbol{x} \\ \sigma\left(W_1 \boldsymbol{x}\right) \end{bmatrix}, \qquad \boldsymbol{\phi}_2\left(\boldsymbol{h}_1(\boldsymbol{x})\right) = \begin{bmatrix} \boldsymbol{h}_1(\boldsymbol{x}) \\ \sigma\left(W_2 \boldsymbol{\phi}_1(\boldsymbol{x})\right) \end{bmatrix}. \tag{2.54}$$

As the output of g must be of the same dimension as the input, the final transformation a final linear transformation W_{m+1} maps the high-dimensional embedding back to the d-dimensional space. We refer to the number of DenseBlocks as the depth of the i-DenseNet, and to the number of added dimensions as the growth of the i-DenseNet.

It can be shown that the Lipschitz constant of a general concatenation of two function is upper bounded as follows:

$$\phi(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{bmatrix} \qquad \mathrm{Lip}(\phi) \leq \sqrt{\mathrm{Lip}(f_1)^2 + \mathrm{Lip}(f_2)^2}. \tag{2.55}$$

As f_1 and f_2 are in this case neural networks constrained to a Lipschitz constant of 1, the Lipschitz constant of the concatenation is the upper bound by $\sqrt{2}$. Dividing by this then enforces $\mathrm{Lip}(\phi) \leq 1$.

In addition, Perugachi-Diaz et al. (2021) introduce the so-called *concatenated LipSwish* (CLipSwish) activation function, which is based on LipSwish activations (R. T. Chen et al., 2019). LipSwish is a slight variation of the Swish activation that ensures $\mathrm{Lip}(\mathrm{LipSwish}) = 1$. The elementwise formulation of LipSwish is given by

$$\mathrm{LipSwish}(x) = \frac{\mathrm{Swish}(x)}{1.1} = \frac{1}{1.1}\left(x \frac{1}{1+e^{-x}}\right). \tag{2.56}$$

CLipSwish further adds concatenations to avoid the small gradients of LipSwish for large negative inputs:

$$\sigma(\boldsymbol{x}) := \mathrm{CLipSwish}(\mathrm{x}) = \begin{bmatrix} \mathrm{LipSwish}(\mathrm{x})/1.004 \\ \mathrm{LipSwish}(-\mathrm{x})/1.004 \end{bmatrix} = \begin{bmatrix} \sigma_1(x) \\ \sigma_2(x) \end{bmatrix}, \tag{2.57}$$

where the factor $1/1.004$ again ensures $\mathrm{Lip}(\mathrm{CLipSwish}) = 1$. This factor is obtained via

$$\mathrm{Lip}(\sigma) = \sup_{\boldsymbol{x}} \| J\sigma(\boldsymbol{x}) \|_2, \tag{2.58}$$

which corresponds to the largest singular value of $J\sigma$, i. e.. the square root of the largest eigenvalue of $(J\sigma(\boldsymbol{x}))(J\sigma(\boldsymbol{x}))^\top$. The matrix $(J\sigma(\boldsymbol{x}))(J\sigma(\boldsymbol{x}))^\top$ turns out to be diagonal, as both of the concatenated functions in σ are elementwise transformations. Consequently, the largest eigenvalue is given by the largest entry of $(J\sigma(\boldsymbol{x}))(J\sigma(\boldsymbol{x}))^\top$:

$$\operatorname{Lip}(\sigma) = \sup_x \max_j \sqrt{\lambda_j} = \sup_x \max_j \sqrt{\frac{\partial \sigma_{1,j}}{\partial x_j}^2 + \frac{\partial \sigma_{2,j}}{\partial x_j}^2}. \tag{2.59}$$

The authors of Perugachi-Diaz et al. (2021) state that the value of 1.004 for CLipSwish was empirically estimated.

2.4.2 *Continuous Normalizing Flows*

Analogous to Normalizing Flows, the bijections obtained by numerically solving Neural-ODEs can be used to flexibly parameterize probability distributions (R. T. Q. Chen et al., 2018). Such models are referred to as continuous Normalizing Flows (CNFs).

In the generative direction, a CNF can be described as follows. Let $\mathbf{Z} \sim p_{\mathbf{Z}}$ be a known d-dimensional base distribution (e.g. $p_{\mathbf{Z}} = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$) with a sample denoted as $\boldsymbol{z} \in \mathbb{R}^d$. In addition, let $\boldsymbol{g} : [t_0, T] \times \mathbb{R}^d \mapsto \mathbb{R}^d$ be a neural network. A continuous Normalizing Flow then defines a continuum of probability densities $\mathbf{X}_t \sim p_{\mathbf{X}_t}$ for $0 \leq t_0 < T$ as the solutions to the initial value problem

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t} \ln p_{\mathbf{X}_t}(\boldsymbol{h}(t, \boldsymbol{z})) = -\operatorname{tr}\left(\dfrac{\partial}{\partial \boldsymbol{h}}\, \boldsymbol{g}\big(t, \boldsymbol{h}(t, \boldsymbol{z})\big)\right) & t \in [t_0, T) \\ \ln p_{\mathbf{X}_0}(0, \boldsymbol{h}(0, \boldsymbol{z})) = \ln p_{\mathbf{Z}}(\boldsymbol{h}(0, \boldsymbol{z})), & \boldsymbol{z} \in \mathbb{R}^d, \end{cases} \tag{2.60}$$

where $\boldsymbol{h} : [t_0, T] \times \mathbb{R}^d \mapsto \mathbb{R}^d$ is the intermediate output of a neural ODE and as such a solution to the IVP

$$\begin{cases} \partial_t \boldsymbol{h}(t, \boldsymbol{z}) = \boldsymbol{g}(t, \boldsymbol{h}(t, \boldsymbol{z})), & t \in [t_0, T), \\ \boldsymbol{h}(0, \boldsymbol{z}) = \boldsymbol{z}, & \boldsymbol{z} \in \mathbb{R}^d. \end{cases} \tag{2.61}$$

Although a continuum of probability densities is parameterized, common probability density estimation tasks often only care about the final density $p_{\mathbf{X}_T}(\boldsymbol{x})$ for an arbitrarily chosen $T \in \mathbb{R}_{>0}$, which can for example be used to approximate the data distribution for generative modeling tasks.

For evaluating the density $p_{\mathbf{X}_T}(\boldsymbol{x})$, Eq. (2.60) has to be solved backwards in time ($\boldsymbol{x} = \boldsymbol{x}_T \to \boldsymbol{z}$), such that the initial density at the *departure* point, i. e.

$p_{\mathbf{Z}}(\boldsymbol{z})$, as well as the density changes along the trajectory can be evaluated. For sampling from a CNF, Eq. (2.61) is to be solved in forward direction with the initial condition $\boldsymbol{z}$ being a random sample from the known base distribution $p_{\mathbf{Z}}$.

The main limitation of CNF-based methods is the computational cost of evaluating the input derivative of a network potentially hundreds of times in an adaptive ODE solver. As a possible remedy (Finlay et al., 2020; Onken et al., 2021) suggest vector field regularizations motivated by optimal transport theory. Specifically, straight trajectories are enforced via transport penalties, simplifying the dynamics for the numerical solver, and consequently enabling cheaper gradient computations during training. Other follow-up work considers the use of the Hutchinson trace estimator (Hutchinson, 1989) for faster divergence calculations (Grathwohl et al., 2019).

In our implementation of the SLDA in Chapter 4 we estimate the divergence exactly by brute force in 2D settings. In 3D settings we make use of the Hutchinson trace estimator similar to Grathwohl et al. (2019). For evaluating the density at multiple different time points in parallel, it is also necessary to solve multiple ODEs in parallel with different time intervals. To do so, we leverage the trick provided in Appendix F of R. T. Chen et al. (2020), which essentially reparameterizes the time of each system by a dummy variable such that we may globally integrate from 0 to 1.

2.4.3 *Conditional Normalizing Flows*

A parameterization for conditional distributions $p_{\mathbf{X}}(\boldsymbol{x}|\boldsymbol{c})$ can be obtained by additionally conditioning the parameters of $\mathcal{T}$ on another variable $\boldsymbol{c} \in \mathbb{R}^{d_c}$ through a hypernetwork (Ha et al., 2017). That is, the bijective layers $\mathcal{T}(\boldsymbol{x})$ are simply exchanged with conditional bijective layers $\mathcal{T}(\boldsymbol{x}, \boldsymbol{c})$ with $\boldsymbol{c}$ being the condition. This is commonly called a conditional Normalizing Flow (Ardizzone et al., 2019; Kobyzev et al., 2020). Conditioning bijective layers is often straightforward, and the process is analogous for a wide range of layers. For the hypernetwork that maps the condition $\boldsymbol{c}$ to the parameters of $\mathcal{T}$ we use residual neural networks, with the number of layers and hidden units being tunable hyperparameters.

2.4.3.1 *Simple Conditional Transforms.*

We illustrate the idea of conditional layers on two simple affine layers. The generalization to other layers is often straight-forward.

Orthogonal Layers. Conditional orthogonal layers can be constructed by conditioning the invertible Householder transformations. Let $\boldsymbol{c} \in \mathbb{R}^{d_c}$ denote the conditioning value. Then a conditional Householder transform is given by

$$\mathcal{T}_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{c}) = H(\boldsymbol{c})\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{v}(\boldsymbol{c})\left(\boldsymbol{v}(\boldsymbol{c})^{\top}\boldsymbol{x}\right) \tag{2.62}$$

$$\mathcal{T}_{\mathrm{H}}^{-1}(\boldsymbol{z}, \boldsymbol{c}) = H(\boldsymbol{c})^{-1}\boldsymbol{z} = \left(I_d - 2\boldsymbol{v}(\boldsymbol{c})\boldsymbol{v}(\boldsymbol{c})^{\top}\right)^{-1}\boldsymbol{z}, \tag{2.63}$$

where $\boldsymbol{v}(\boldsymbol{c})$ has length 1, i. e. $\boldsymbol{v}(\boldsymbol{c}) = \frac{\boldsymbol{v}^*(\boldsymbol{c})}{||\boldsymbol{v}^*(\boldsymbol{c})||_2}$, and $\boldsymbol{v}^*(\boldsymbol{c}) : \mathbb{R}^{d_c} \mapsto \mathbb{R}^d$ is a neural network.

The Jacobian determinant remains the same as in the unconditional setting:

$$|\det(J\mathcal{T}_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{c})| = 1. \tag{2.64}$$

SVD Layers. Analogous to the unconditional SVD layers, the transformation is given by

$$\mathcal{T}_{\mathrm{SVD}}(\boldsymbol{x}, \boldsymbol{c}) = U(\boldsymbol{c})S(\boldsymbol{c})V(\boldsymbol{c})^{\top}\boldsymbol{x} \tag{2.65}$$

$$\mathcal{T}_{\mathrm{SVD}}^{-1}(\boldsymbol{z}, \boldsymbol{c}) = V(\boldsymbol{c})S(\boldsymbol{c})^{-1}U(\boldsymbol{c})^{\top}\boldsymbol{z}. \tag{2.66}$$

The orthogonal transformations are again implemented via d consecutive Householder transformations, but this time conditioned on $\boldsymbol{c}$. The diagonal entries of S are given by a neural network followed by an exponential transform to ensure positivity.

The determinant is given by

$$|\det(J\mathcal{T}_{\mathrm{SVD}}(\boldsymbol{x}, \boldsymbol{c}))| = |\det(S(\boldsymbol{c}))| = \prod_{i=1}^{d}[S(\boldsymbol{c})]_{ii}. \tag{2.67}$$

2.4.3.2 *Conditional Invertible Residual Networks*

While many bijective layers are straightforward to condition, invertible residual networks pose an exception. For i-ResNets and thus i-DenseNets it is not possible to directly parameterize the matrix weights by another neural network.

Recall from Section 2.4.1.2 that a Lipschitz constant smaller than 1 has to be enforced for i-ResNets. In the unconditional setting, this is done by spectral normalization, i.e. the estimation of the spectral norm and subsequent projection of the matrix weights (Behrmann et al., 2019; Perugachi-Diaz et al., 2021). If the weights are not freely learnable parameters, but instead outputs of a hyper-network, this is not directly applicable.

To avoid this issue, we propose to directly pass the condition $\boldsymbol{c} \in \mathbb{R}^{d_c}$ (or an embedding of this condition) as an additional input to the i-ResNet, effectively conditioning the first layer. Let $\boldsymbol{g} : \mathbb{R}^{d+d_c} \mapsto \mathbb{R}^d$ be a layer with augmented input and Lipschitz constant $K < 1$. It is then straightforward to show that for a fixed condition $\boldsymbol{c}$ the function $\boldsymbol{g_c} : \mathbb{R}^d \mapsto \mathbb{R}^d$ also has a Lipschitz constant of at most K.

Theorem 3. *Let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{d_{\text{total}}}$ be vectors such that $\boldsymbol{x} = \begin{bmatrix} x_{\text{in}} \\ x_{\text{cond}} \end{bmatrix}$, with $\boldsymbol{x}_{\text{in}} \in \mathbb{R}^{d_{\text{in}}}$ and $\boldsymbol{x}_{\text{cond}} \in \mathbb{R}^{d_{\text{cond}}}$, and let $d_{\text{total}} = d_{\text{in}} + d_{\text{cond}}$. Further, let $\boldsymbol{f} : \mathbb{R}^{d_{\text{total}}} \mapsto \mathbb{R}^d$ be a Lipschitz continuous function with Lipschitz constant K, i. e.*

$$\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{f}(\boldsymbol{y})\|_2 \leq K\|\boldsymbol{x} - \boldsymbol{y}\|_2 \qquad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^{d_{\text{total}}}. \tag{2.68}$$

Then $\boldsymbol{f_c} : \mathbb{R}^d \mapsto \mathbb{R}^d$ with $\boldsymbol{f_c} := \boldsymbol{f}\left(\begin{bmatrix} x_{\text{in}} \\ \boldsymbol{c} \end{bmatrix}\right)$ for a fixed $\boldsymbol{c} \in \mathbb{R}^{d_{\text{cond}}}$ has a Lipschitz constant of at most K.

Proof.

$$\begin{aligned} &||\boldsymbol{f}\left(\begin{bmatrix} x_{\text{in}} \\ x_{\text{cond}} \end{bmatrix}\right) - \boldsymbol{f}\left(\begin{bmatrix} y_{\text{in}} \\ y_{\text{cond}} \end{bmatrix}\right)||_2 \leq K\|\begin{bmatrix} x_{\text{in}} \\ x_{\text{cond}} \end{bmatrix} - \begin{bmatrix} y_{\text{in}} \\ y_{\text{cond}} \end{bmatrix}\|_2 \\ \Rightarrow &||\boldsymbol{f}\left(\begin{bmatrix} x_{\text{in}} \\ \boldsymbol{c} \end{bmatrix}\right) - \boldsymbol{f}\left(\begin{bmatrix} y_{\text{in}} \\ \boldsymbol{c} \end{bmatrix}\right)||_2 \leq K\|\begin{bmatrix} x_{\text{in}} \\ \boldsymbol{c} \end{bmatrix} - \begin{bmatrix} y_{\text{in}} \\ \boldsymbol{c} \end{bmatrix}\|_2 = K\|\boldsymbol{x}_{\text{in}} - \boldsymbol{y}_{\text{in}}\|_2 \\ \Rightarrow &||\boldsymbol{f_c}(\boldsymbol{x}_{\text{in}}) - \boldsymbol{f_c}(\boldsymbol{y}_{\text{in}})||_2 \leq K\|\boldsymbol{x}_{\text{in}} - \boldsymbol{y}_{\text{in}}\|_2 \end{aligned} \tag{2.69}$$

□

Conditional i-ResNets. Let g be a neural network with $\text{Lip}(g) < 1$. A conditional i-ResNet and i-DenseNet layer can then be constructed with

$$\mathcal{T}_{\text{RES}}(\boldsymbol{x}, \boldsymbol{c}) = \boldsymbol{x} + \boldsymbol{g}\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{c} \end{bmatrix}\right) \tag{2.70}$$

$$\mathcal{T}_{\text{RES}}^{-1}(\boldsymbol{y}, \boldsymbol{c}) = \lim_{m\to\infty} \boldsymbol{h}^{(m)}. \tag{2.71}$$

where the inverse is obtained as the fix point solution to

$$\boldsymbol{h}^{(0)} = \boldsymbol{y}, \qquad \boldsymbol{h}^{(m+1)} = \boldsymbol{y} - \boldsymbol{g}\left(\begin{bmatrix} \boldsymbol{h}^{(m)} \\ \boldsymbol{c} \end{bmatrix}\right). \tag{2.72}$$

We are not aware of any other use of such conditional layers in the literature, but note that this kind of conditioning for i-ResNets was mentioned, but not used or explored further, by Biloš et al. (2021).

2.5 PHYSICS INFORMED NEURAL NETWORKS

Remark. *In this section we provide an introduction to Physics-informed Neural networks (PINNs), a general framework for enforcing PDEs as constraints in neural networks. This introduction mainly serves as the background for the particle-density PINNs proposed in Chapter 3. We note that parts of this section are based on the related work section of our first contribution Arend Torres et al. (2022).*

Despite the relevant progress in numerical PDE solvers, a seamless incorporation of data remains an open problem (Freitag, 2020; Kalnay et al., 2007). To fill this gap, physics-informed neural networks (PINNs) have emerged as an attractive alternative to classical methods for data-based forward and inverse solving of PDEs. The general idea of PINNs is to use the expressive power of modern neural architectures to solve PDEs in a data-driven way; see (Raissi et al., 2019). To do so, a PDE-based penalty is introduced in the optimization of the neural network. By minimizing this penalty on points sampled throughout the signal domain, the PDE is (weakly) enforced. As the evaluation of this PDE-penalty is based on automatic differentiation, PINNs are straightforward to implement in both forward and inverse settings.

Consider time-dependent parameterized PDEs of the general form

$$\boldsymbol{f}(t, \boldsymbol{x}|\boldsymbol{\lambda}) := \partial_t \boldsymbol{u}(t, \boldsymbol{x}) + \mathcal{P}(\boldsymbol{u}|\boldsymbol{\lambda}) = \mathbf{0}, \tag{2.73}$$

where $\boldsymbol{u}$ is a (possibly vector-valued) function of interest, $\mathcal{P}$ is a non-linear operator parameterized by $\boldsymbol{\lambda}$, and ∂_t is the partial time derivative w.r.t. time $t \in [0, T]$. The position $\boldsymbol{x} \in \Omega$ is defined in a spatial domain $\Omega \subseteq \mathbb{R}^d$. The PDE is subject to initial condition $\boldsymbol{g}_{\text{ic}}$ and boundary conditions $g_{\partial\Omega}$

$$\boldsymbol{u}(0, \boldsymbol{x}) = \boldsymbol{g}_{\text{ic}}(\boldsymbol{x}), \qquad \boldsymbol{u}(t, \boldsymbol{x}) = g_{\partial\Omega}(\boldsymbol{x}), \tag{2.74}$$

for $\boldsymbol{x} \in \partial\Omega$ and $t \in [0, T]$. In classical forward problems, the interest lies in simulating $\boldsymbol{u}$ in a setting where it is fully specified, i. e. with fully known initial conditions, boundary conditions, and $\boldsymbol{\lambda}$.

Although applicable in well-posed settings, PINNs mainly shine in ill-posed settings with some information missing. That is, the initial conditions might be unknown, the boundary conditions might be unknown, and even the PDE parameters $\boldsymbol{\lambda}$ might be unknown, giving the problem an inverse aspect. Instead, only a small set of N noisy observations $\boldsymbol{u}_{\text{obs}}$

$$\boldsymbol{u}(t^{(i)}, \boldsymbol{x}^{(i)}) + \boldsymbol{\epsilon}^{(i)} = \boldsymbol{u}_{\text{obs}}^{(i)} \tag{2.75}$$

with noise

$$||\boldsymbol{\epsilon}^{(i)}||_2^2 \ll ||\boldsymbol{u}^{(i)}||_2^2 \qquad \forall i \in \{0, 1, \dots, N\} \tag{2.76}$$

is given.

The goal of PINNs is to approximate $\boldsymbol{u}(t, \boldsymbol{x})$ with a neural network. In addition to directly fitting the network to the observations, the PDE Eq. (2.73) is enforced by minimizing a penalty. Possibly known initial or boundary conditions are generally enforced by augmenting the dataset with noise-free observations, although recent work recommends hard-encoding boundary conditions in the network (Sukumar & Srivastava, 2021).

2.5.1 *Parameterization and Loss Functions*

In a PINN, $\boldsymbol{u}(t, \boldsymbol{x})$ (and thus $\boldsymbol{f}(t, \boldsymbol{x})$) is approximated by the network $\boldsymbol{u}_\theta(t, \boldsymbol{x})$. Parameters θ are adjusted by minimizing the reconstruction loss for the available observations (L_{obs}), and adding a penalty term for each additional constraint available. That is, (i) to enforce the PDE in the domain (L_f), and (ii) for the boundary conditions (L_b) and (iii) for the initial conditions (L_i):

$$\theta = \underset{\theta \in \Theta}{\arg\min} \left[L_{\text{obs}}(\mathcal{D}, \theta) + w_1 L_{\text{pde}}(\theta) + w_2 L_{\text{bc}}(\theta) + w_3 L_{\text{ic}}(\theta) \right], \tag{2.77}$$

with loss weights $w_j \in \mathbb{R}_{\geq 0}$, and training data $\mathcal{D} = (\boldsymbol{t}_{obs}, \boldsymbol{X}_{obs}, \boldsymbol{U}_{obs})$. Each loss term works as a penalizer to weakly enforce the given constraint, be that a boundary condition or the PDE itself.

Let $\ell(\hat{y}, y)$ be a non-negative loss function, such as, for example, the squared Euclidean distance $\ell(\hat{y}, y) = ||\hat{y} - y||_2^2$. Then, similar to the formulation of Sirignano and Spiliopoulos (2018), individual losses commonly employed in PINNs are given by or proportional to:

$$L_{\text{obs}}(X, \boldsymbol{t}, \boldsymbol{u}_{obs}, \theta) = \frac{1}{N} \sum_{i=0}^{N} \ell(u_\theta(t^{(i)}, \boldsymbol{x}^{(i)}), u_{obs}^{(i)}), \tag{2.78}$$

$$L_{\text{pde}}(\theta) = \frac{1}{|[0, T] \times \Omega|} \int_0^T \int_\Omega \|f_\theta\,(t, \boldsymbol{x}|\boldsymbol{\lambda})\|_2^2 \,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t, \tag{2.79}$$

$$L_{\text{bc}}(\theta) = \int_0^T \int_{\partial\Omega} \ell\Big(\boldsymbol{u}_\theta(t, \boldsymbol{x}), g_{\partial\Omega}(\boldsymbol{x})\Big) \,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t, \tag{2.80}$$

$$L_{\text{ic}}(\theta) = \int_{\partial\Omega} \ell\left(\boldsymbol{u}_\theta(0, \boldsymbol{x}), g_{\text{ic}}(\boldsymbol{x})\right) \mathrm{d}\boldsymbol{x}. \tag{2.81}$$

For sufficiently high weights and a sufficiently flexible neural network, the losses enforce the PDE and other constraints within the domain. In practice, the integrals are approximated by points sampled within the signal domain or its boundary. Of particular interest are the so-called collocation points $\{t_{\text{pde}}^{(i)}, \boldsymbol{x}_{\text{pde}}^{(i)}\}_{i=0}^{N_{\text{pde}}}$ selected to approximate L_{pde}:

$$L_{\text{pde}}(\theta) \approx \frac{1}{N_{\text{pde}}} \sum_{i=0}^{N_{\text{pde}}} \| f_\theta \left(t_{\text{pde}}^{(i)}, \boldsymbol{x}_{\text{pde}}^{(i)} | \boldsymbol{\lambda} \right) \|_2^2. \tag{2.82}$$

Initial approaches for selecting the collocation points in PINNs relied on a fixed grid (Lagaris et al., 1998; 2000; Rudd, 2013). Follow-up work proposed stochastic estimates of the integral via (quasi-) Monte Carlo methods (J. Chen et al., 2019; L. Lu et al., 2021; Sirignano & Spiliopoulos, 2018) or Latin Hypercube sampling (Raissi et al., 2019). In general, since only a discrete number of collocation points can be selected for the continuous domain, resampling these collocation points during training can be performed to cover (in expectation) the entire domain.

2.5.2 *Application Settings*

PINNs can be easily adapted to multiple different settings without requiring significant changes to the method. We mainly consider two important settings:

1. If $\boldsymbol{\lambda}$ is known, the PDE is fully specified and the objective is to find a solution $\boldsymbol{u}$ in a data-driven manner by training a neural network. The PDE takes the role of a regularizer, where the particular physical laws provide the prior information. This is similar to classical data assimilation settings. Applications range widely, from spatiotemporal models for turbulent flows in fluid dynamics (Angriman et al., 2023) to compartmental models in epidemiology (Berkhahn & Ehrhardt, 2022; Ning et al., 2023).

2. If $\boldsymbol{\lambda}$ is (partially) unknown, an inverse aspect is added to the problem setting. That is, the interest lies in inferring $\boldsymbol{\lambda}$ from observations by including them in the optimization process. For example, physical properties such as the viscosity coefficient of a fluid might be learned (Jagtap et al., 2020).

As a simple example for both settings, consider the movements of a pendulum, where the angle $u(t)$ is a function of time and governed by the equation

$$f(t|\lambda) := \frac{d^2u(t)}{dt^2} + \boldsymbol{\lambda} \sin(u(t)).$$

In the first settings, very few noisy observations of angles at different time points are given and the parameter λ is known. A vanilla neural network easily overfits the problem, as shown in Fig. 2.5. By additionally regularizing the network with a PDE penalty, the generalization of the network is substantially improved, as shown in Fig. 2.6.

If λ is also unknown as in the second setting, it can still be inferred from the data with a PINN. The parameter λ is initialized with a guess and left as a freely learnable parameter. By training the network with a PDE loss, the λ parameter can be recovered given sufficient data, as shown in Fig. 2.7.

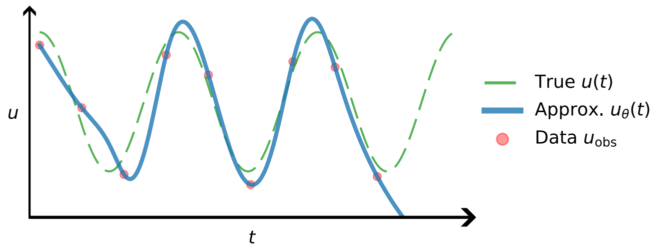


Figure 2.5: Approximation of the angle of a pendulum by a neural network given noisy measurements. Without any regularization, the network overfits the scarce and noisy data.

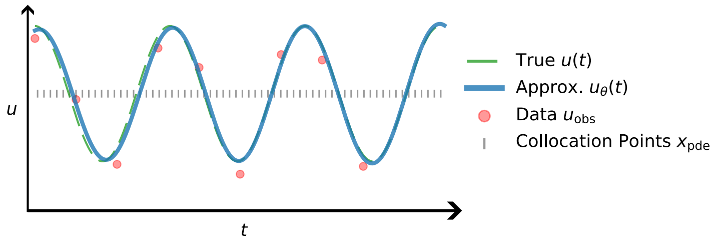


Figure 2.6: Approximate solution to the equation of motion of a pendulum by a PINN given noisy measurements and the PDE $f(t|\lambda) := \frac{d^2u(t)}{dt^2} + \lambda \sin(u(t))$. Collocation points were selected quasi-randomly.

2.5.3 *Early Work and Origin*

Solving PDEs minimizing penalties in neural networks was, to the best of our knowledge, pioneered in the 1990s by Dissanayake and Phan-Thien

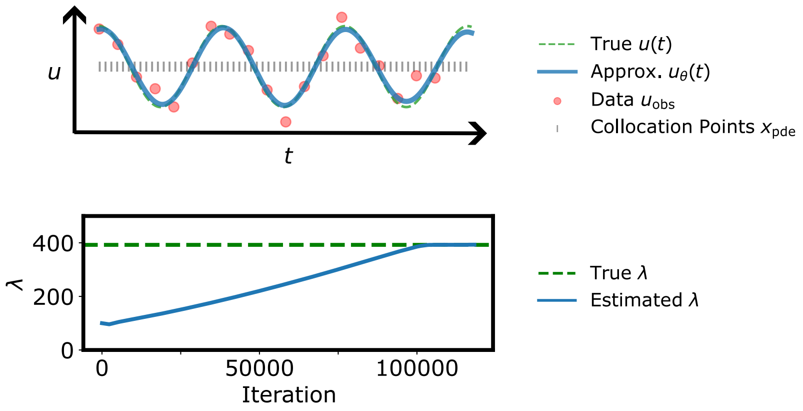


FIGURE 2.7: Inverse learning of the parameter λ given noisy measurements of pendulum angles with a PINN. Collocation points were selected quasi-randomly. Top: The learned function $u_\theta(t)$ given noisy observations. Bottom: The estimated parameter λ as a function of training iterations.

(1994) and van Milligen et al. (1995), which focused on time-independent settings. Later adoptions such as Parisi et al. (2003) extended it to non-steady and time-dependent settings.

These models were repopularized with the work of Raissi et al. (2017) and Raissi et al. (2019) that extended the concept to a wide range of non-linear time-dependent PDEs and introduced the term *physics-informed neural network*. The *deep Galerkin method* (DGM) (Sirignano & Spiliopoulos, 2018) reformulated the grid-based losses in PINNs as integrals over the signal domain that are estimated with uniformly sampled collocation points. This sampling of collocation points was claimed as a mesh-free version of PINNs. In addition, Sirignano and Spiliopoulos (2018) provide a formal theoretical background for PINNs, providing a convergence theorem for a class of quasilinear parabolic PDE. This theorem is partly based on universal approximation theorems for neural networks.

2.5.4 *Flexible Activations for Low-Dimensional Input*

Although neural networks are universal function approximators, in practice they struggle to learn high-frequency signals from low-dimensional input. Recent work discusses that fully connected networks with ReLU or tanh activations have a spectral bias, i. e. they prioritize learning low-frequency modes (Rahaman et al., 2019; Wang et al., 2022; Xu et al., 2019). To overcome this problem, concepts such as positional encoding (Mildenhall et al., 2020), random Fourier features (Tancik et al., 2020), and sinusoidal activations (Sitzmann et al., 2020) have been proposed.

The sinusoidal activations presented by Sitzmann et al. (2020) essentially substitute commonly used activation functions such as the ReLU with periodic non-linear activations, such that the transformation at layer j is given by

$$\boldsymbol{g}_j(\boldsymbol{x}) := \sin(\omega_j W_j(\boldsymbol{x}) + \boldsymbol{b}_j). \tag{2.83}$$

Sitzmann et al. (2020) propose $\omega_0 = 30$ and $\omega_j = 1$ for $j > 0$, and additionally propose an initialization scheme that preserves the distribution of the activation functions throughout the network. The resulting sinusoidal representation networks (SIREN) show much higher flexibility than classical activations for learning high-frequency signals in continuous representations of 2D images, signed distance functions in 3D, and for solving differential equations with PINNs.

3 MESH-FREE EULERIAN PINNS

Remark. *This chapter closely follows the work in Arend Torres et al. (2022).*

Physics-informed Neural Networks (PINNs) have recently emerged as a principled way to include prior physical knowledge in the form of partial differential equations (PDEs) into neural networks. By enforcing a PDE based loss on data points sampled in the signal domain, PINNs provide a simple but still widely applicable framework to combine observations with prior physical knowledge. As PINNs do not introduce any sort of mesh, but rely on neural networks that are continuous in the signal domain, they are often referred to as a *mesh-free* approach. This contrasts with the more classical mesh-based approaches, such as finite-difference methods. However, the enforcement of the PDE loss still relies on the discretization of space by choosing collocation points. These points are usually selected uniformly within a bounded region, even in settings with spatially sparse signals. Furthermore, if the boundaries are not known, the selection of such a region is difficult and often results in a large proportion of collocation points being selected in areas of low relevance. To resolve this drawback of current methods, we present a mesh-free and adaptive approach termed particle-density PINN (pdPINN), which is inspired by the microscopic viewpoint of fluid dynamics. The method is based on the Eulerian viewpoint and, unlike classical mesh-free methods, does not require the introduction of Lagrangian updates. We propose sampling directly from the distribution over the particle positions, eliminating the need to introduce boundaries while adaptively focusing on the most relevant regions. This is achieved by interpreting a non-negative physical quantity (such as the density or temperature) as an unnormalized probability distribution from which we sample with dynamic Monte Carlo methods. The proposed method leads to higher sample efficiency and improved performance of PINNs. These advantages are demonstrated in various experiments based on continuity equations, Fokker-Planck equations, and the heat equation.

3.1 PHYSICS-INFORMED NEURAL NETWORKS

As discussed in the previous background Section 2.5, the fundamental idea of PINNs is to regularize neural networks by enforcing partial differential equations within the signal domain. Recalling the setting, we consider time-dependent parameterized PDEs of the general form

$$\boldsymbol{f}(t, \boldsymbol{x}|\boldsymbol{\lambda}) := \partial_t \boldsymbol{u}(t, \boldsymbol{x}) + \mathcal{P}(\boldsymbol{u}|\boldsymbol{\lambda}) = \mathbf{0}, \tag{3.1}$$

where $\boldsymbol{u}$ is a (possibly vector-valued) function of interest, $\mathcal{P}$ is a non-linear operator parameterized by $\boldsymbol{\lambda}$, and ∂_t is the partial time derivative w.r.t. time $t \in [0, T]$. The position $\boldsymbol{x} \in \Omega$ is defined in a spatial domain $\Omega \subseteq \mathbb{R}^d$. The PDE is subject to initial condition $\boldsymbol{g}_{\text{ic}}$ and boundary conditions $g_{\partial\Omega}$

$$\boldsymbol{u}(0, \boldsymbol{x}) = \boldsymbol{g}_{\text{ic}}(\boldsymbol{x}), \qquad \boldsymbol{u}(t, \boldsymbol{x}) = g_{\partial\Omega}(\boldsymbol{x}), \tag{3.2}$$

for $\boldsymbol{x} \in \partial\Omega$ and $t \in [0, T]$. However, instead of full knowledge of the λ, initial, and boundary conditions, we are given a small set of N noisy observations $\boldsymbol{u}_{\text{obs}}$

$$\boldsymbol{u}(t^{(i)}, \boldsymbol{x}^{(i)}) + \epsilon^{(i)} = \boldsymbol{u}_{\text{obs}}^{(i)} \tag{3.3}$$

with noise $\epsilon^{(i)} \ll \boldsymbol{u}^{(i)}\ \forall i \in \{0, 1, \ldots, N\}$.

PINNs then approximate $\boldsymbol{u}(t, \boldsymbol{x})$, and subsequently $\boldsymbol{f}(t, \boldsymbol{x})$, by the network $\boldsymbol{u}_\theta(t, \boldsymbol{x})$. The parameters θ are adjusted by minimizing the combined loss of (i) reconstructing available observations (L_{obs}), (ii) softly enforcing the PDE constraints on the domain (L_{pde}), and (iii) fulfilling the boundary (L_{bc}) and initial conditions (L_{ic}). A common choice for individual losses is the expected L^2 loss, approximated via the average L^2 loss over the observations and via sampled boundary and initial conditions, respectively.

3.1.1 *Problem Setting*

PINNs rely on neural networks that are continuous in both space and time and are commonly referred to as mesh-free, since no discretization of the function approximation is needed. However, this ignores the way PINNs are actually trained in practice. That is, the PDE loss L_{pde} requires a similar discretization step to approximate an integral over the continuous signal domain, that is

$$L_{\text{pde}}(\theta) = \frac{1}{|[0,T] \times \Omega|} \int_{t=0}^{T} \int_{\Omega} ||f_\theta(t, \boldsymbol{x})||_2^2 \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t = E_{p(t,\boldsymbol{x})}\left[||f_\theta(t,\boldsymbol{x})||_2^2\right] \approx \frac{1}{n}\sum_{i=1}^{n} ||f_\theta(t_i, \boldsymbol{x}_i)||_2^2 \tag{3.4}$$

with $p(t, \boldsymbol{x}) \propto 1$ being supported on $[0, T] \times \Omega$. While Sirignano and Spiliopoulos (2018) argue that a randomized sampling of the collocation points qualifies as being mesh-free, an important and valued property of such methods is missing. That is, mesh-free methods are usually able to adapt according to the evolution of the PDE.

Although there are alternative sampling methods as discussed in Section 2.5, all of these rely on uniform proposal distributions and cannot be directly applied if there are no known boundaries or boundary conditions, e.g. for $\Omega = \mathbb{R}^d$. In addition, problems can arise if the constrained region is large compared to the area of interest. Considering, for example, the shock wave (of a compressible gas) in a comparably large space, most collocation points would fall into areas of low density. We argue that, due to the locality of particle interactions, regions with higher density are more relevant for regularizing the network. In addition, if the quantity of interest is a density, it is natural to focus on such a high-density region.

To address these shortcomings of previous methods, we propose a mesh-free and adaptive approach for sampling collocation points, illustrated on the example of compressible fluids. By changing $p(t, \boldsymbol{x})$ to the distribution over the particle positions in the fluid we effectively change the loss functional in Eq. (3.4). We then generalize to other settings, such as thermodynamics, by interpreting a positive, scalar quantity of interest with a finite integral as a particle density. Within this work, we specifically focus on PDEs that can be derived based on local particle interactions or can be shown to be equivalent to such a view, as, for example, is the case for the heat equation with its connection to particle diffusion. Notably, we do not require the introduction of Lagrangian updates, as classical mesh-free methods do. Such a Lagrangian view would instead be based on evaluating the PDE with respect to moving parcels or particles.

Our main contributions are as follows:

- We demonstrate that PINNs with uniform sampling strategies fail in settings with spatially sparse signals as well as in unbounded signal domains; these problems can severely degrade the network's predictive performance.

- To overcome these limitations of existing approaches, we propose a truly mesh-free version of PINNs, in which the collocation points are sampled using physics-motivated MCMC methods. By staying within the Eulerian framework, we avoid conceptual challenges of classical mesh-free methods based on Lagrangian updates, such as the enforcement of boundary conditions.
- The proposed model is applicable to a wide range of dynamical systems governed by PDEs that share an underlying microscopic particle description, such as several hydrodynamic, electro- and thermodynamic problems.
- We evaluate and compare our proposed method with existing approaches in up to 3-dimensional settings. Compared to existing mesh refinement methods, significantly fewer collocation points are required to achieve similar or better predictive performances.

3.2 RELATED WORK

3.2.1 *Mesh-Free Fluid Dynamics*

Classical mesh-free approaches in computational fluid dynamics are based on non-parametric function representations. Smoothed Particle Hydrodynamics (SPH) (Gingold & Monaghan, 1977; Lind et al., 2020) are the most prominent example. In SPH, fluid properties such as the density and pressure are represented by a discrete set of particles and interpolated using a smoothing kernel function. For updating the function forward in time, the particles have to be propagated according to the Lagrangian formulation of the PDE, relying on the kernel to compute the spatial derivatives. One of the benefits of such a representation is that the mass is conserved by construction. However, Lagrangian updates become challenging when enforcing boundary conditions, requiring the introduction of ad-hoc "dummy" or "mirror" particles (Lind et al., 2020). Instead, we present a mesh-free, particle-based, PINN that does not require Lagrangian updates, and it is already applicable in the Eulerian formulation. It should be noted that the proposed pdPINNs can in principle be combined with Lagrangian updates, such as those proposed by Raissi et al. (2019) and later by Wessels et al.

(2020). But as the intention of this work is to improve upon current Eulerian PINNs, we refer to future work for the comparison and extension to the Lagrangian formalism.

3.2.2 *Alternative Meshes and Losses for PINNs*

Recent work proposes local refinement methods for PINNs by adding more samples within regions of high error (L. Lu et al., 2021; Tadiparthi & Bhattacharya, 2021). Residual adaptive refinement (RAR) is suggested by L. Lu et al. (2021), which is based on regularly evaluating the PDE loss on a set of uniformly drawn samples. The locations corresponding to the highest PDE loss are then added to the set of collocation points used in training. Tadiparthi and Bhattacharya (2021) further enhance RAR by learning a linear map between the uniform distribution and the distribution over the PDE loss by optimizing an optimal transport objective. By sampling uniformly and subsequently transforming these samples, it is attempted to focus on regions of higher error. Due to the conceptual similarity to RAR, we will denote this method as OT-RAR. The work of Nabian et al. (2021) explores *Importance Sampling* based on the (unnormalized) proposal distribution $||f_\theta(t,x)||_2^2$ for a more sample efficient evaluation of Eq. (3.4). Samples are drawn by discretizing the signal domain.

However, in all these cases the underlying mechanism for exploring regions of high error is based on (quasi-) uniform sampling within the boundaries. As such, they do not resolve the issues of unknown boundaries and, furthermore, will be infeasible in higher dimensions.

3.3 PARTICLE-DENSITY PINNS

In this section we introduce the concept of mesh-free *particle-density PINNs (pdPINNs)*. First, we examine the limitations of the common PDE loss in Eq. (3.4) and, secondly, we present a solution by integrating over the position of particles instead of the full support of the signal domain.

The underlying assumption of our approach is that the dynamics described by the PDE can be explained in terms of local interactions of particles. This is the case, for instance, for commonly considered dynamics of gases, liquids or active particles (Hoover & Hoover, 2003; Toner & Tu, 1995), and specifically for PDEs that are based on the continuity equation for the conservation of mass.

3.3.1 *Existing limitations of Eulerian PINNs*

Consider the problem of modeling a (possibly non-steady) compressible fluid, i.e. a fluid with a spatially and temporally evolving density $\rho(t, \boldsymbol{x})$ and velocity $\boldsymbol{v}(t, \boldsymbol{x})$. For the sake of notational brevity, we will denote these by ρ and $\boldsymbol{v}$. Given noisy observations, our particular interest lies in the prediction of particle movements, hence in the approximation of the density (and potentially other physical quantities) with a neural network ρ_θ. Additional quantities such as the velocity or pressure might also be observed and modeled.

Commonly, the PDE then serves as a physics-based regularizer of the network by enforcing the PDE loss L_{pde} in Eq. (3.4) during standard PINN training. For this, L_{pde} is evaluated on a set of collocation points that are, for example, uniformly distributed on a bounded region. However, the limitations of this approach already become apparent when considering a simple 1D advection problem with constant velocity defined by the following PDE:

$$\partial_t \rho + \boldsymbol{v} \cdot (\nabla \rho) = 0. \tag{3.5}$$

Figure 3.1 illustrates a one-dimensional case on the domain $[0, T] \times \Omega$, with $\Omega = \mathbb{R}$, and a known constant velocity $v \propto 1$. We measure the density $\rho^{(i)}$ at different (spatially fixed) points in time and space $\{(t^{(i)}, \boldsymbol{x}^{(i)})\}$, on which a neural network $\rho_\theta(t, \boldsymbol{x})$ is trained. For optimizing the standard PDE loss L_{pde} as given in Eq. (3.4), we would require a bounded region $\Omega_{\mathcal{B}} := [a, b] \subset \Omega$ with $a < b$ and $a, b \in \mathbb{R}$. This, in turn, leads to two issues:

1. Since the moving density occupies a small subset of Ω, uniformly distributed collocation points within $\Omega_{\mathcal{B}}$ will enforce Eq. (3.5) in areas with low-density. This results in insufficient regularization of ρ_θ.

2. Defining a suitable bounded region $\Omega_{\mathcal{B}}$ requires a priori knowledge about the solution of the PDE, which is generally not available. Choosing too tight boundaries would lead to large parts of the density moving out of the considered area $\Omega_{\mathcal{B}}$. Too large boundaries would instead lead to poor regularization as this would worsen the sparsity problem in issue (1.).

In practice, most Eulerian PINNs approaches opt for naively defining a sufficiently wide region $\Omega_{\mathcal{B}}$, resulting in a poor reconstruction. In the context of our advection problem, this is showcased in Figure 3.1b. To properly resolve the aforementioned issues, one should (i) focus on areas

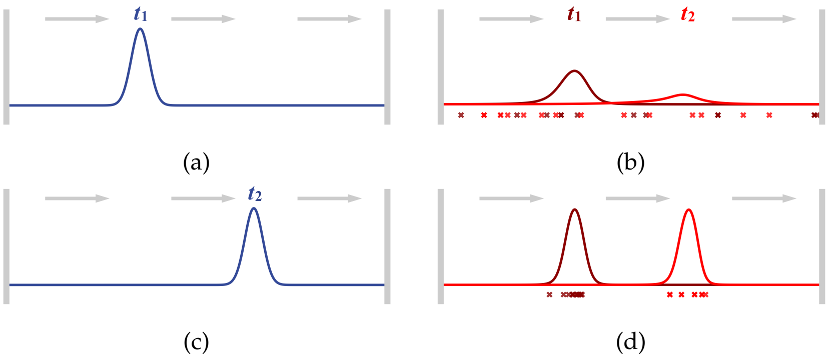


FIGURE 3.1: Advection experiment in 1D: (a) ground truth at time t_1 and (c) time t_2, (b) density prediction with uniform collocation points and (d) particle-density-based collocation points for $t \in \{t_1, t_2\}$, with crosses indicating sampled points.

that have a relevant regularizing effect on the prediction of ρ_θ and (ii) adapt to the fluid movements without being restricted to a predefined mesh.

3.3.2 *Mesh-Free Eulerian PINNs*

We thus propose to reformulate the PDE loss in Eq. (3.4) as the expectation of $||f_\theta(t, \boldsymbol{x})||_2^2$ with respect to the molecular distribution $\Psi(t, \boldsymbol{x})$

$$L_{\text{pd}}(\theta) \approx \int_{t=0}^{T} \int_{\Omega} \Psi(t, \boldsymbol{x}) \left[||f_\theta(t, \boldsymbol{x})||_2^2 \right] \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t. \tag{3.6}$$

The *molecular distribution function* $\Psi(t, \boldsymbol{x}, \boldsymbol{v})$ originates from kinetic theory, where it is defined such that

$$\int_{\Delta \boldsymbol{x}} \int_{\Delta \boldsymbol{u}} \Psi(t, \boldsymbol{x}, \boldsymbol{u}) \, \mathrm{d}\boldsymbol{u} \, \mathrm{d}\boldsymbol{x} \tag{3.7}$$

is the probability that a molecule with a velocity within $\Delta \boldsymbol{u} = \Delta u_1 \Delta u_2 \Delta u_3$ occupies the volume $\Delta \boldsymbol{x} = \Delta x_1 \Delta x_2 \Delta x_3$.

This completely removes the need of defining ad hoc boundaries while providing the ability to flexibly focus on highly relevant regions, i.e. those that are more densely populated. As the particle density corresponds directly to the occupation probability of a molecule $\Psi(t, \boldsymbol{x})$ with a changed normalization constant, we can estimate L_{pd} via samples drawn from the normalized particle density, which is denoted as ρ_N. For homogeneous fluids, this coincides with the normalized mass density.

In summary, we propose to draw collocation points from the normalized density:

$$(t_i, \boldsymbol{x}_i) \text{ drawn from } (\mathbf{T}, \mathbf{X}) \sim \rho_N(t, \boldsymbol{x}) = \tfrac{1}{Z}\rho(t, \boldsymbol{x}). \tag{3.8}$$

The true particle positions and the density ρ_N are however unknown in practice. Instead, we have to rely on the learned density $\rho_\theta(t, \boldsymbol{x})$ as a proxy provided by the neural network. We denote the associated normalized PDF by $q_\theta(t, \boldsymbol{x}) = \frac{1}{Z'}\rho_\theta(t, \boldsymbol{x})$ with support on $[0, T] \times \Omega$. The PDE loss is then defined as the expectation w.r.t. $q_\theta(t, \boldsymbol{x})$:

$$L_{\text{pdpinn}}(\theta) = \mathbb{E}_{q_\theta(t,\boldsymbol{x})}\left[||f_\theta(t, \boldsymbol{x})||_2^2\right] \tag{3.9}$$

$$= \int_{t=0}^{T} \int_{\Omega} q_\theta(t, \boldsymbol{x}) \; ||f_\theta(\boldsymbol{x}, t)||_2^2 \; \mathrm{d}\boldsymbol{x} \; \mathrm{d}t. \tag{3.10}$$

In order to approximate this integral, samples need to be drawn from $q_\theta(t, \boldsymbol{x})$. This can be done in a principled way by using dynamic Monte Carlo methods, despite the fact that the normalization constant Z is unknown. We highlight that, in contrast to the mesh-based loss in Eq. (3.4), the loss in Eq. (3.9) is also suitable for problems on unbounded domains such as $\Omega = \mathbb{R}^d$.

3.3.3 *Applicability of pdPINNs*

Although motivated in the context of an advection problem, the proposed approach is generally applicable to a wide range of PDEs. The advection equation 3.5 can be seen as a special case of mass conservation (assuming $\nabla \cdot \boldsymbol{v} = 0$), which is one of the fundamental physical principles expressed as a *continuity equation*. This continuity equation relates temporal changes of the fluid density ρ to spatial changes of the flux density $\rho\boldsymbol{v}$ through

$$\partial_t \rho + \nabla \cdot (\rho \boldsymbol{v}) = 0. \tag{3.11}$$

Another common physical process that is suited for our approach is diffusion, such as in the Heat Equation, where local interactions of particles give rise to the following PDE (as established by Fick's second law):

$$\partial_t T - \alpha \nabla^2 T = 0, \tag{3.12}$$

where T denotes the temperature interpreted as density, α the thermal (or mass) diffusivity, and ∇^2 the Laplacian operator. By introducing additional constraints to the mass-conservation, one can describe viscous fluids with the Navier-Stokes equations or even self-propelled, active particles, for which Toner and Tu (Toner & Tu, 1995; 1998; Tu et al., 1998) introduced hydrodynamic equations. Other possible applications involve Maxwell's equations for conservation of charge in electrodynamics, as well as the distribution of Brownian particles with drift described by the Fokker-Planck equations. In general, our method is applicable in settings where (i) a non-negative scalar field (with a finite integral) of interest can be interpreted as a particle density, and (ii) the local interactions of these particles give rise to the considered PDEs.

3.4 MODEL AND IMPLEMENTATION

A wide range of different network architectures and optimization strategies for PINNs have emerged. They emphasize well-behaved derivatives with respect to the input domain (Sitzmann et al., 2020), allow higher expressivity for modeling high frequency data (Tancik et al., 2020; Wang, Wang, & Perdikaris, 2021), or resolve gradient pathologies within PINNs (Wang, Teng, & Perdikaris, 2021). As our method does not rely on a specific architecture, any such improvement can be easily combined with the proposed pdPINNs. For the experiments in this work we will use simple fully-connected networks with sinusoidal (Sitzmann et al., 2020) or tanh activations. Code is available in the supplementary material [1].

3.4.1 *Finite total density*

For reformulating the predicted density ρ_θ as a probability, we have to ensure non-negativity as well as a finite integral over the input domain Ω. Non-negativity can for example be achieved via a squared activation function after the last layer. An additional bounded activation function g is then added, which guarantees the output to be within a pre-specified range $[0, c_{max}]$. The integral $\mathbb{R}^d$ can then be enforced to be finite by multiplying the bounded output with a Gaussian kernel. Summarizing these three steps, let $\tilde{\rho}_\theta$ denote the output of the last layer of our fully connected neural network and $p_{\text{gauss}}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \mu, \Sigma)$, then we predict the density ρ_θ as

$$\rho_\theta(t, \boldsymbol{x}) = p_{\text{gauss}}(\boldsymbol{x})\; g(\tilde{\rho}_\theta(t, \boldsymbol{x})^2) \leq c_{\max} p_{\text{gauss}}(\boldsymbol{x}). \tag{3.13}$$

1 https://openreview.net/attachment?id=253DOGs6EF&name=supplementary_material.

In practice, the choice of $c_{\max}$ does not affect the model as long as it is sufficiently large. The used mean μ and covariance Σ are maximum likelihood estimates based on the observations $\boldsymbol{X}_{\text{obs}}$, i. e. the sample mean $\bar{\boldsymbol{x}}$ and covariance $\bar{\Sigma}$ of the sensor locations. To allow more flexibility in the network, we add a scaled identity matrix to the covariance $\Sigma = \bar{\Sigma} + c \cdot I$.

3.4.2 *Background Sampling for pdPINNs*

At initialization, the network prediction ρ_θ is random and thus does not carry any useful information, i. e. sampling from this density would be meaningless. Therefore, we start training the pdPINNs with a warm-up phase in which samples are obtained from a prespecified background distribution:

$$(\mathbf{T}_{bg}, \mathbf{X}_{bg}) \sim p_{\text{bg}}(t, \boldsymbol{x}) = p(t) p_{\text{bg}}(\boldsymbol{x}|t) \tag{3.14}$$

with $p(t) = \mathcal{U}(0, T)$. For $p_{\text{bg}}(\boldsymbol{x}|t)$ we use random linear combinations of the convex hull of $\{x^{(i)}\}_{i=1}^{N}$ spanned by c data points summarized as rows of the matrix $Z \in \mathbb{R}^{c \times d}$. This leads to $\boldsymbol{x} = \boldsymbol{m}Z$ with weight $\boldsymbol{m} \in \mathbb{R}^c$ which can be drawn from a Dirichlet distribution, i. e. $\boldsymbol{m} \sim \text{Dir}(\boldsymbol{\alpha} = \mathbf{1})$. Of course, a uniform sampling mechanism on a defined region is also suitable, if for example the boundary of the domain is known.

We initially draw all samples from the background distribution and then slowly increase the proportion of samples obtained from the particle density, as we found that leaving some background samples slightly helps in the training.

3.4.3 *Markov chain Monte Carlo (MCMC) sampling*

Finally, MCMC methods allow us to draw samples from the unnormalized density $\rho_\theta(t, \boldsymbol{x})$. We consider several MCMC samplers and emphasize that the wide range of well-established methods offers the ability to use a specialized sampler for the problem considered, if the need may arise. Gradient-based samplers such as Hamiltonian Monte Carlo (Betancourt, 2017; Duane et al., 1987) are particularly suited for our setting, as the gradients of ρ_θ with respect to the input space are readily available. For problems where boundaries are known and we have to sample from a constrained region, a bijective transformation is used so that the Markov chain may operate in an unconstrained space (Parno & Marzouk, 2018). In our experience, both Metropolis Hastings and Hamiltonian Monte Carlo already worked sufficiently well for a wide range of PDEs without requiring much fine-tuning. We highlight that pdPINNs do not directly depend on MCMC as a sampler, and alternative sampling methods such as modern variational inference schemes (Rezende & Mohamed, 2015) can also be used directly as a substitute.

3.5 EXPERIMENTS

In this section we demonstrate the advantages of pdPINNs compared to *uniform sampling*, *importance sampling* (Nabian et al., 2021) as well as the adaptive refinement methods *RAR* (L. Lu et al., 2021) and *OT-RAR* (Tadiparthi & Bhattacharya, 2021). Despite the term *uniform* sampling, we rely in all our experiments on quasi-random Sobol sequences for more stable behavior in the low samples regime. To guarantee a fair comparison, we considered slight variations of the proposed implementations of RAR and OT-RAR, so that only a limited number of collocation points are used. For the pdPINNs we consider multiple MCMC schemes, including Metropolis-Hastings (MH-pdPINN), and Hamiltonian Monte Carlo (HMC-pdPINN) methods. In addition, we consider a sampling scheme based on the discretization of the domain similar to the one used in Nabian et al. (2021). Due to its similarity to inverse transform sampling in 1D, we refer to it as IT-pdPINN.

The models in sections 3.5.1 and 3.5.2 are implemented in PyTorch (Paszke et al., 2019), with a custom Python implementation of the Monte Carlo samplers. For the Fokker-Planck experiment in section 3.5.3, we make use of the efficient MCMC implementations provided by TensorFlow probability (Abadi et al., 2016b; Lao et al., 2020) and the utilities of the DeepXDE library (L. Lu et al., 2021). Table 3.1 provides an overview of the different experiment setting, and the used architectures.

3.5.1 *Continuity equation for simulated particles*

As a challenging prediction task we consider a setting motivated by the real world problem of modelling bird densities and velocities measured from a set of weather radars (Dokter et al., 2011; Nussbaumer et al., 2019; 2021) – or more generally the area of radar aeroecology. A non-steady compressible fluid in three dimensions is simulated by propagating particles through a

TABLE 3.1: Architecture overview for the different experiments.

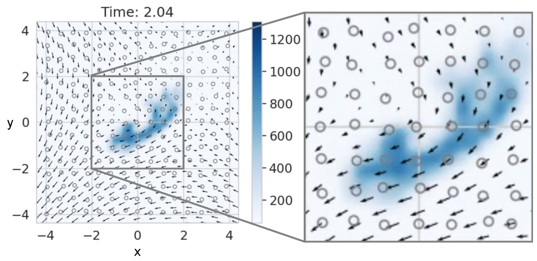
Experiment	Input	Output	Layers	Units	σ
Continuity Eq. (2D)	$[0, T] \times \mathbb{R}^2$	$\rho_\theta \in \mathbb{R}_+$	2	256	sin
		$v_\theta \in \mathbb{R}^2$	1	64	sin
Continuity Eq. (3D)	$[0, T] \times \mathbb{R}^3$	$\rho_\theta \in \mathbb{R}_+$	6	256	sin
		$v_\theta \in \mathbb{R}^3$	3	256	sin
Heat Eq.	$[0, T] \times \mathbb{R}^2$	$T \in \mathbb{R}_+$	2	32	tanh
Fokker-Planck Eq.	$[0, T] \times \mathbb{R}$	$p_\theta \in \mathbb{R}_+$	5	64	sin

pre-defined velocity field, i.e. the fluid is simulated using the continuity equation as the underlying PDE (see Eq. (3.11)):

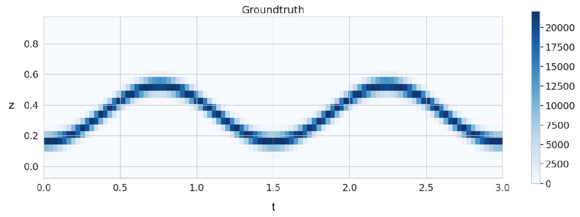
$$\partial_t \rho + \nabla \cdot (\rho v) = 0. \tag{3.15}$$

Data generation. To provide the network with training observations, we introduce a set of spatially fixed sensors (comparable to *radars*) which count over time the number of fluid particles within a radius r and over 21 contiguous altitude layers. Another disjoint set of sensors is provided for the validation set while the test performance is evaluated on a grid. The birds-eye view of the setting is shown in Figure 3.2a, where circles indicate the area covered by the radars. Figure 3.2b additionally shows the 3D simulated data projected along the z-axis and over time. In the Appendix section A.2.1 we describe the data generation and training setting in detail and provide the corresponding code in the supplementary. We evaluate pdPINNs both in complete 3D setting, as well as in a simplified 2D setting where we discard the z-axis.

Implementation. For modeling the density and velocity, two sinusoidal representation networks (SIREN) (Sitzmann et al., 2020) $\rho_\theta(t, \boldsymbol{x})$ and $v_\theta(t, \boldsymbol{x})$ are used, which are then regularized by enforcing the continuity equation for the conservation of mass (see Eq. (3.11)). To showcase the sample efficiency of pdPINNs, experiments are performed over a wide range of collocation points (256 to 65536 in 3D, and 16 to 8192 in 2D). In each setting, the weights for the PDE loss were selected using a grid search with repeated



(a)



(b)

FIGURE 3.2: Visualization of the compressible fluid experiment. (a) Bird-eye view of the ground truth particle density. (b) z-projection of the density over time, obtained by summing over the xy grid cells.

random runs. As metric we used the 1st quartile of the validation set R^2, which discourages settings that are unstable over multiple runs.

Results in 3D. The resulting box plots of test R^2 are provided in Figure 3.3, where the *Baseline* corresponds to training without any PDE loss. The proposed pdPINN approach clearly outperforms alternative (re-)sampling methods across all numbers of collocation points. Already with very few collocation points (512) pdPINNs achieve results that require orders of magnitude more points (32768) for uniform sampling. Finally, we observe that the performance gap shrinks as the number of collocation points increases, eventually converging to the same limiting value. Even when getting close to the memory limit of a NVIDIA Titan X GPU, other sampling strategies at best achieve comparable results with pdPINNs.

Figure 3.4 showcases the projection of the density onto the z axis for the OT-RAR method and the Metropolis-Hastings based pdPINN. A random seed and 2048 collocation points were used. The OT-RAR PINN shows disconnected density predictions that clearly violate mass conservation, whereas the Metropolis Hastings based pdPINN is capable of mostly preserving it.

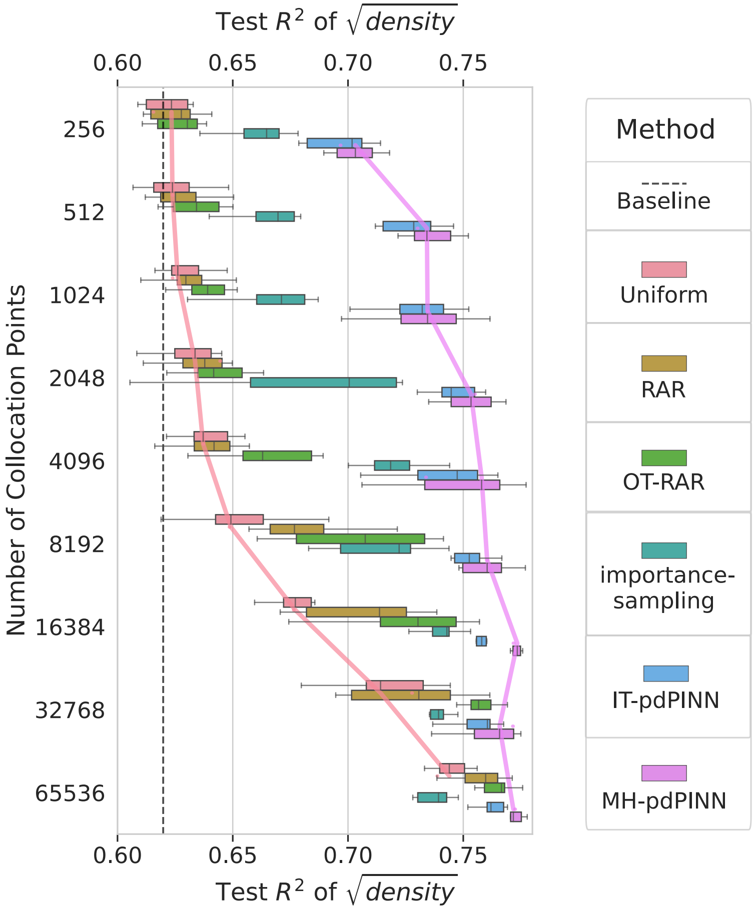


FIGURE 3.3: Explained variance of $\sqrt{\rho}$ evaluated on the test set, for different number of collocation points for the 3D mass conservation experiment and for 10 different seeds.

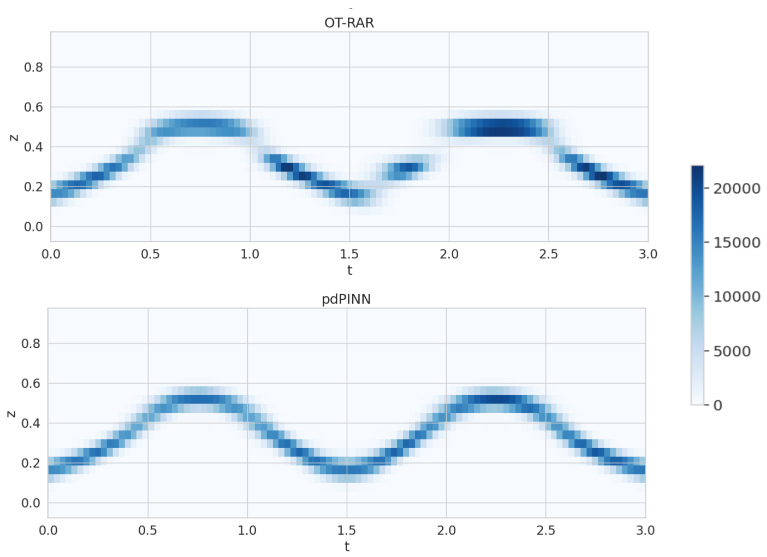


FIGURE 3.4: Mass conservation experiment (3D): Predictions (obtained with 2048 collocation points) summed over xy grid cells to obtain z-axis projection over time. Top: OT-RAR. Bottom: MH-pdDPINN.

Results in 2D. As an additional experiment, we simplified the setting by projecting the data onto the xy-axis, i.e. the birds-eye view, which is a common setting for geostatistical data (e.g. in Nussbaumer et al. (2019)). The general setup is similar to the 3D setting, although a smaller network is used (see Table 3.1). The results in this 2D setting are shown in Figure 3.5. The results are comparable in nature to the 3D setting, although with a smaller performance gap with respect to alternative sampling methods. This decrease of the gap is to be expected, as the lower dimensional space is much easier to explore with uniform proposals.

3.5.2 *Heat Equation*

We further consider a 2D diffusion problem where randomly distributed sensors provide measurements of the temperature. More specifically, the dynamics of the data are given by the heat equation

$$\partial_t T - \alpha \nabla^2 T = 0. \tag{3.16}$$

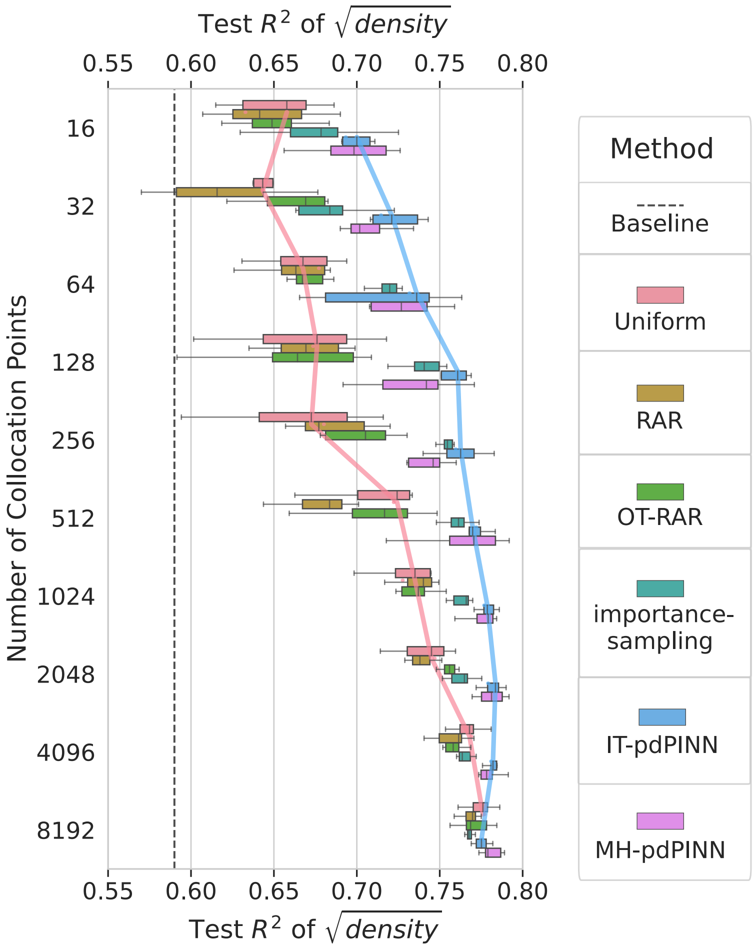


FIGURE 3.5: Explained variance of $\sqrt{\rho}$ evaluated on the test set, for different number of collocation points for the 2D mass conservation experiment.

Data generation. We focus on a general setting with the initial conditions being zero temperature everywhere except for a specified region, as shown in Figure 3.6a, and we let the system evolve for $t \in [0, 0.2]$. The dataset was generated by numerically solving the heat equation through the finite difference method, precisely the Forward Time, Centered Space (FTCS) approximation (Recktenwald, 2004). The PINN is only provided sensor measurements of the temperature.

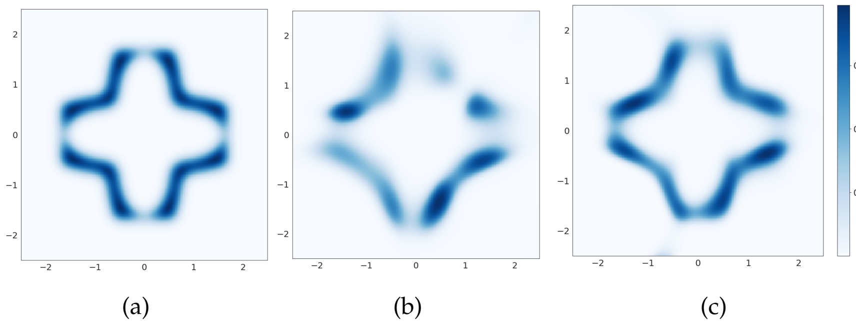


FIGURE 3.6: Temperature predictions of the heat equation experiment (trained with 128 collocation points) at time $t \sim 0.044$. (a) Ground truth (b) uniform sampling, and (c) pdPINN.

Results. Temperature predictions for PINNs with uniform sampling and pdPINNs are illustrated in Figure 3.6b and 3.6c, respectively, with the ground truth in Figure 3.6a. We can observe that the uniform sampling strategy does not allow to focus on the relevant parts of the domain, i.e. regions with high temperature, and that it visibly fails to reconstruct the temperature profile. In contrast, the pdPINN promotes sampling in regions of higher density and predicts the true temperature more reliably.

Figure 3.7 illustrates the test R^2 of the predicted T averaged over 20 different seeds. Error bars correspond to the 95% confidence interval for the mean estimate, based on 1000 bootstrap samples, while the colors indicate the different PDE weights w_2 explored. As in previous settings, we show that with few samples (16) the regularization enforced by the PDE loss is not strong enough, leading to comparable results in both approaches (as expected). Hence, PINNs and pdPINNs show similar results in this regime. However, as the number of samples increases (32-64-128-256), the PDE loss enforced by the proposed pdPINNs quickly and steadily outperforms uniform sampling. Lastly, we also verified that in the limit of high samples (512-1024) the two sampling strategies converge, as in such a low-dimensional domain the uniform samples fully and densely cover the considered area. This, again, is in line with the observed results of the other experiments.

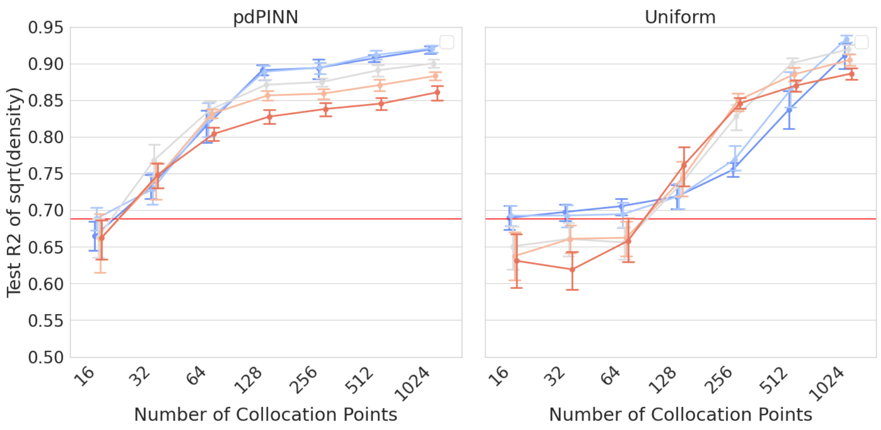


FIGURE 3.7: Test R^2 of predicted T in the heat equation experiment as a function of different number of collocation points. Results are averaged over 20 different seeds and the resulting error bars correspond to 95% confidence interval for the mean estimation, based on 1000 bootstrap samples. Different colors indicate different PDE weights w_2.

3.5.3 *Fokker-Planck Equation*

For a demonstration of a setting without any observed data but only initial conditions, we solve the Fokker-Planck (FP) equations in a setting where an analytical solution is available (cf. Särkkä and Solin (2019)). The FP equations describe the evolution of the probability density of the movement of Brownian particles under a drift. More specifically, assume we are given particles at time t_0, which are distributed according to $p(t_0, x)$. Let the movements of these particles be described by the following stochastic differential equation, where W_t denotes the standard Wiener process:

$$dX_t = \mu(t, X_t)\, dt + \sigma(t, X_t)\, dW_t \tag{3.17}$$

with known drift $\mu(X_t, t)$ and diffusion coefficient $D(X_t, t) = \sigma^2(X_t, t)/2$. The FP equation for the probability density $p(t, x)$ of the random variable X_t is then given by

$$\frac{\partial}{\partial t} p(t, x) + \frac{\partial}{\partial x}\left[\mu(t, x) p(t, x)\right] - \frac{\partial^2}{\partial x^2}\left[D(t, x) p(t, x)\right] = 0. \tag{3.18}$$

We train a network to predict the (probability) density $p_\theta(t, x)$. Data is only provided for the initial condition, and the PDE loss is based on

Eq. (3.18) within space $\Omega = [-.1.5, 1.5]$ and time $t \in [-1, 1]$. As the analytical solution is available in form of a probability density, we can estimate the KL divergence $KL(p||p_\theta)$ to evaluate the performance. Furthermore, we can sample collocation points from the true particle distribution $p(t, x)$ (referred to as "$p(t, x)$ as sampler"), offering a "best case scenario" of pdPINNs. A total of 5000 collocation points were used and the weights were manually adjusted based on the error on a validation set. The number of steps was fixed at 30000 for all methods.

Data Generation. We consider the following setting over the time interval $[t_0, t_n] = [-1, 1]$ with drift function μ, noise σ and initial particle positions $p(x|t = t_0)$ given by

$$\mu(X_t, t) = \mu(t) = \sin(10t) \tag{3.19}$$

$$\sigma(X_t, t) = \sigma = 0.06 \tag{3.20}$$

$$p(x|t = t_0) = \mathcal{N}(0, 0.02^2 \cdot I_d) \tag{3.21}$$

The Fokker-Planck equation then has an analytical solution (cf. Särkkä and Solin, 2019) which is given by

$$p(x|t) = \mathcal{N}(\mu_s(t), \sigma_s^2(t)) \tag{3.22}$$

$$p(t) = \mathcal{U}(t_0, t_n) \tag{3.23}$$

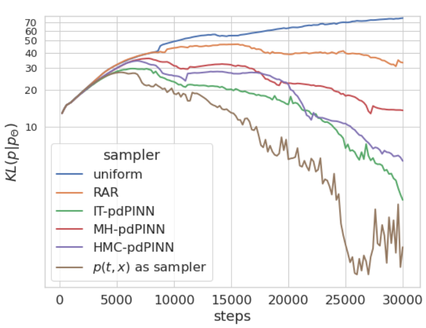
$$\mu_s(t) = -\frac{\cos(10t)}{10} + \frac{\cos(10)}{10} \tag{3.24}$$

$$\sigma_s^2(t) = 0.0036t + 0.004. \tag{3.25}$$

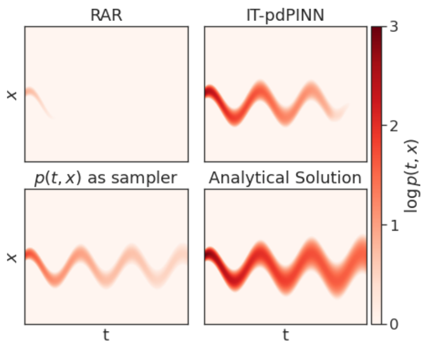
For evaluating the deviation of our prediction to the solution, we evaluate the KL divergence between the analytical solution and the network approximation $KL(p(x, t)|\hat{p}_\Theta(x, t))$ by sampling 10000 points from the true $p(x, t)$.

Results. Figure 3.8a shows the evolution of KL divergence during training on a log-scale, highlighting that pdPINN based methods require fewer steps to achieve a low divergence. In addition, sampling from the true particle distribution leads to the fastest improvement and the lowest divergence after 30000 training steps. A qualitative comparison of the results is given in Figure 3.8b, showing that RAR and uniform sampling fail to propagate the sine wave forward.

Walltime of different MCMC samplers. Within the first two experiments we relied on custom implementations of the MCMC samplers in PyTorch



(a)



(b)

FIGURE 3.8: Fokker-Planck equation in 1D. (a) KL divergence between the true target distribution and approximation during training, (b) predicted $\log p_\theta(t, x)$ after training, cropped to $x \in [-0.5, 0.5]$.

that were not optimized with respect to computational performance. For the Fokker-Planck experiment we provide an additional implementation in Tensorflow (probabilitiy). This implementation provides a realistic wall-time estimate by relying on highly efficient MCMC implementations. More specifically, we compare the following methods for selecting collocation points:

(i) Uniform sampling

(ii) Resampling based on Residual Adaptive Refinement (RAR) (L. Lu et al., 2021)

(iii) pdPINN based on a discrete approximation of the target distribution (Tadiparthi & Bhattacharya, 2021) (IT-pdPINN)

(iv) pdPINN with Metropolis-Hastings (MH) MC with parallel tempering (Earl & Deem, 2005)

(v) pdPINN with Hamiltonian MC (HMC) with parallel tempering (Earl & Deem, 2005) and dual averaging step-size adaptation (Hoffman, Gelman, et al., 2014, Section 3.2)

The wall times for the different methods in 1D are provided in Figure 3.9. Wall times were evaluated on a NVIDIA GeForce RTX 2080 Ti and an Intel(R) Xeon(R) CPU E5-1660 v3 @ 3.00GHz processor. It can be seen that the *IT-pdPINN* method based on a discretized approximation achieves a similar speed to uniform sampling. Thus, in low dimensions and with known boundaries, IT-pdPINNs provide a computationally cheap alternative. In higher dimensions, or with unknown boundaries, it might be necessary to rely on MCMC based methods such as MH or HMC. Although Metropolis-Hastings and Hamiltonian Monte Carlo require more time per step, the proposals of pdPINN only require the evaluation of the density and are thus not dependent on the PDE at hand. Other resampling schemes rely on evaluating the PDE loss on proposal points, which can be expensive in memory and compute if higher order derivatives are required.

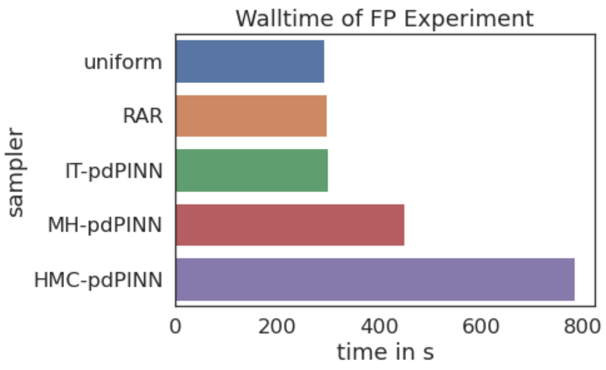


FIGURE 3.9: Total run-times for the Fokker-Planck experiment with fixed number of iterations. Seeds were selected randomly.

3.6 CONCLUSION

In this work, we introduced a general extension to PINNs applicable to a variety of problem settings that involve physics-based regularization of neural networks. These range from hydrodynamic flow problems to electro- and thermo-dynamic problems, as well as more general applications of the Fokker-Planck equations.To overcome the limitations of classical mesh-based Eulerian PINNs, we introduce a novel PDE loss that is defined with respect to the particle density in rather general types of PDEs. By employing MCMC methods to sample collocation points from the density approximated by the network, we derive an efficient and easy-to-implement improvement for providing a more appropriate regularization objective in PINNs. In particular, our new pdPINNs are completely mesh-free, thereby overcoming the severe efficiency problems of classical PINNs in high-dimensional and sparse settings. Further, the absence of a mesh allows us to elegantly handle settings with uncertain or unknown domain boundaries.

4 LAGRANGIAN FLOW NETWORKS

Remark. *This chapter closely follows the work in Torres et al. (2024) with some additional unpublished work mostly restricted to additional experiments for evaluating the sinusoidal activations for invertible densenets.*

We introduce *Lagrangian Flow Networks* (LFlows) for modeling fluid densities and velocities continuously in space and time. By construction, the proposed LFlows satisfy the continuity equation, a PDE describing mass conservation in its differential form. Our model is based on the insight that solutions to the continuity equation can be expressed as time-dependent density transformations via differentiable and invertible maps. This follows from classical theory of the existence and uniqueness of Lagrangian flows for smooth vector fields. Hence, we model fluid densities by transforming a base density with parameterized diffeomorphisms conditioned on time. The key benefit compared to methods relying on numerical ODE solvers or PINNs is that the analytic expression of the velocity is always consistent with changes in density. Furthermore, we require neither expensive numerical solvers, nor additional penalties to enforce the PDE. LFlows show higher predictive accuracy in density modeling tasks compared to competing models in 2D and 3D, while being computationally efficient. As a real-world application, we model bird migration based on sparse weather radar measurements.

4.1 MOTIVATION AND SETTING

The development of physics-informed Machine Learning (PI-ML) (Karniadakis et al., 2021) opens new opportunities to combine the power of modern ML methods with physical constraints that serve as meaningful regularizers. These constraints might, for example, be available in the form of partial differential equations (PDEs). Within PI-ML we consider hydrodynamic flow problems governed by the physical law of mass conservation.

This law is described in its local and differentiable form by a PDE commonly known as the *continuity equation* (CE)

$$\begin{cases} \partial_t \rho + \nabla \cdot (\boldsymbol{v}\rho) = 0 & (t, \boldsymbol{x}) \in (t_0, T) \times \Omega, \\ \rho(t_0, \boldsymbol{x}) = \rho_{t_0}(\boldsymbol{x}) & \boldsymbol{x} \in \Omega. \end{cases} \tag{4.1}$$

For any time $t \in [t_0, T)$ the function $\rho(t, \cdot)$ can be thought of as the density of parcels advected by the velocity field $\boldsymbol{v}$, with initial density ρ_{t_0}. Here, $[t_0, T] \times \Omega$ is the space-time domain, which is a subset of $\mathbb{R} \times \mathbb{R}^d$. The partial derivative w.r.t. time t is denoted by ∂_t and $\nabla \cdot \boldsymbol{b} = \nabla_{\boldsymbol{x}} \cdot \boldsymbol{b}$ is the spatial divergence of a d dimensional vector field $\boldsymbol{b} : [t_0, T] \times \Omega \mapsto \mathbb{R}^d$.

Unlike classical initial value problems, we consider challenging settings where exact boundary and initial conditions are unknown. That is, the continuous density and velocity fields have to be inferred from sparse and noisy data. The important physical constraint is that the solution must comply with the CE in Eq. (4.1). To this end we propose a neural network based model that fulfills the CE by construction and provides physically consistent velocity and density fields.

Two settings. We specifically consider two distinct application settings. In both settings, we are restricted to sparse and noisy measurements. We are further mainly interested in accurately modeling the density, with the velocity measurements or additional equations serving as an informative prior.

In *setting (i)* we measure the fluid density ρ and velocity $\boldsymbol{v}$, without knowing any additional equations other than CE. This occurs, for example, within the area of radar ornithology (Chilson et al., 2017), where the density and velocity of birds can be inferred from radar data. Such radar-based measurements are to date the only practical high-throughput data source for birds. The goal is to spatio-temporally estimate bird migration densities (Nussbaumer et al., 2019). Recent work explores a hydrodynamic view of bird densities for post hoc model analysis (Nussbaumer et al., 2021), modeling densities in discretized domains (Lippert, Kranstauber, Forré, & van Loon, 2022), or in purely simulated data settings (Lippert, Kranstauber, van Loon, & Forré, 2022). With the migration-specific dynamics of birds unknown, the considered hydrodynamic view is synonymous with the continuity equation for the conservation of mass.

In setting (ii) we exclusively have (sparse) density observations, but we know additional equations constraining the velocity. This might, for example, occur in dynamical optimal transport problems. Here, two densities are to be interpolated while adhering to the CE. Therefore, minimizing the total transport cost constrains the velocity field. Other applications may include for example the modeling of animal abundances based on spatiotemporal count data obtain from citizen observations (see e.g. Sullivan et al. (2014)), e.g. counts of specific bird species at a certain location and time. Based on such data Fuentes et al. (2023) recently proposed an approach to model population flows of birds by learning discrete transition probabilities of particles between cells. Aside from matching densities, they encourage a high entropy and penalize customly defined energy (or transport) cost. A continuous generalization of this would correspond to setting (ii). More generally, setting (ii) includes a wide range of compressible fluids dynamic problems.

Main contributions. The main contributions of this paper are as follows:

- We outline a fundamental link between densities modeled by conditional Normalizing Flows and spatiotemporal density fields that satisfy the CE.
- We leverage this link to introduce models for ill-posed hydrodynamic flow problems that always satisfy the CE by construction, coined *Lagrangian Flow Networks* (LFlows). We do so without requiring an explicit representation of the initial density.
- We provide a way to calculate the velocity without inverting the conditional Normalizing Flow, enabling the use of flexible bijective layers with otherwise costly inverses.
- We assess LFlows in multiple application settings and show better predictive performance than existing methods while staying computationally feasible and physically consistent.

4.2 RELATED WORK

Physics-informed Neural Networks Physics-informed neural networks (PINNs) (Raissi et al., 2019) enforce PDEs in neural networks by introducing an additional PDE-loss that penalizes pointwise deviations from the PDE. The PDE-loss is enforced on so-called *collocation points* that are sampled in the signal domain. The accuracy of PINNs is thus limited by the amount (and distribution) of sampled collocation points, as well as the dimension of the signal domain. For conservation laws, recent improvements in PINNs suggest more sophisticated sampling approaches (Arend Torres et al., 2022), or introduce domain decompositions (Jagtap et al., 2020). Although this alleviates scaling problems, the fundamental limitation due to the number of possible collocation points (or subdomains) still remains.

Neural Networks for Conservation Laws. To the best of our knowledge, Richter-Powell et al. (2022) are the first to propose a parameterization of neural networks that enforces mass conservation by design. We refer to the proposed architecture as *Divergence-free Neural Networks* (DFNNs). As the name suggests, solutions to the CE in Eq. (4.1) are represented as divergence-free vector fields in an augmented $(d+1)$ dimensional space. The input $\boldsymbol{s}$ and the vector field are defined as

$$\boldsymbol{b} := \begin{pmatrix} \rho \\ \rho \boldsymbol{v} \end{pmatrix}, \qquad \boldsymbol{s} := \begin{pmatrix} t \\ \boldsymbol{x} \end{pmatrix}. \tag{4.2}$$

The continuity equation can then be rewritten as the divergence of $\boldsymbol{b}$ with respect to the augmented coordinate space $\boldsymbol{s}$

$$\frac{\partial \rho}{\partial t} + \nabla_{\boldsymbol{x}} \cdot (\rho \boldsymbol{v}) = \sum_{i=1}^{d+1} \frac{\partial b_i}{\partial s_i} = \nabla_{\boldsymbol{s}} \cdot \begin{pmatrix} \rho \\ \rho \boldsymbol{v} \end{pmatrix} = \nabla_{\boldsymbol{s}} \cdot \boldsymbol{b} = 0. \tag{4.3}$$

In 3D, a divergence-free vector may be obtained by computing the curl of another vector, the so-called vector potential. The generalization of this approach to higher dimensions is achieved through the concept of differential forms. The resulting parameterization, however, heavily relies on expensive higher-order automatic differentiation, posing limitations in terms of scalability.

Concurrent to our work, L. Li et al. (2023) (TIPF) proposes the use of Lagrangian flow maps to model continuous probability flows that fulfill the CE. Unlike LFlows, TIPF considers well-posed PDE settings with known initial conditions. Specifically, they focus on the Fokker-Planck equations and Wasserstein gradient flows and introduce an unbiased self-consistency loss. In contrast, we consider ill-posed data-assimilation for physical problems with sparse and noisy data where initial conditions are unknown. Additionally, LFlows stand out as they do not require the inversion of bijective layers for computing the velocity, allowing for more expressive bijections.

Physics-Informed Machine Learning. Deep operator learning (Z. Li et al., 2020; L. Lu et al., 2019) was proposed as a general approach to learn dynamics from dense observations. In the considered setting, the PDE is not provided, but has to be learned from large amounts of dense data often provided through simulations. As such, it is not applicable to our setting with spatially sparse data obtained at irregular time steps. Further notable mentions are Lagrangian and Hamiltonian neural networks (Cranmer et al., 2020; Greydanus et al., 2019), which can learn conservation laws from the observed trajectories of individual particles. Sturm and Wexler (2022) propose a photochemistry surrogate model that enforces conservation of atoms between discrete states. A neural network predicts the time-integrated changes in concentration of a discrete box-model, and the conservation law is enforced by an adjustment of the last layer of the network.

Data assimilation with the Adjoint. The continuous adjoint method (Cacuci, 1981a; 1981b; Pontryagin, 1987) allows to differentiate through numerical solvers by integrating adjoint equations backward in time. By minimizing an objective function, it is then possible to infer the initial conditions and parameters of a dynamical system from data. Within adjoint methods, the well-established semi-Lagrangian data assimilation (SLDA) approach is conceptually the closest to our model and setting (Diamantakis & Magnusson, 2016; Robert, 1982; Staniforth & Côté, 1991). SLDA is widely used to integrate transport equations into atmospheric models (Diamantakis, 2013; Hersbach et al., 2020). The basic idea is to evaluate the density at a given sensor location by propagating it back in time to its so-called departure point. This is done by solving the Lagrangian formulation of the continuity equation. The observed density at its departure point and time is then compared to the parameterized initial density to calculate the loss, which is optimized by the adjoint state method.

Relevantly, an efficient autograd implementation of the continuous adjoint state method was presented in the context of neural ODEs by R. T. Q. Chen et al. (2018). The implementation enables black-box differentiation for numerically solved ODEs, and further allows to specify the dynamics of ODEs with a neural network. Interestingly, the simultaneously introduced Continuous Normalizing Flows (CNFs) can be seen as a special case of SLDA. We note that a limiting factor of neural ODE based methods is their computational cost, since the input derivatives of the network that provides the velocity v_θ are repeatedly evaluated for every step of the solver. Furthermore, the dynamics given by neural networks can become stiff during training, leading to problems with adaptive ODE solvers. To avoid these issues, Biloš et al. (2021) propose time-dependent bijections instead of neural ODEs for modeling time series data.

SLDA with neural ODEs. From a fluid dynamics perspective, the solved equations in CNFs correspond to the continuity equation in its Lagrangian formulation, written in terms of the log density, i. e.

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t} \ln \rho(t, \boldsymbol{x}(t, \boldsymbol{z})) = -\nabla \cdot \boldsymbol{v}_\theta(t, \boldsymbol{x}(t, \boldsymbol{z})) \\ \quad \ln \rho(0, \boldsymbol{x}(0, \boldsymbol{z})) = \ln \rho_0(\boldsymbol{x}(0, \boldsymbol{z})). \end{cases} \tag{4.4}$$

with $\boldsymbol{z} \in \mathbb{R}^d$, $\boldsymbol{v}_\theta : [0, T] \times \mathbb{R}^d \mapsto \mathbb{R}^d, t \in [0, T]$, and fixed initial density with unit integral, e.g. $\rho_0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The so-called hidden output of the CNF, which we denote by $\boldsymbol{x}(t, \boldsymbol{z})$, can be interpreted as the position of a moving parcel as it is a solution to the IVP

$$\begin{cases} \partial_t \boldsymbol{x}(t, \boldsymbol{z}) = \boldsymbol{v}_\theta(t, \boldsymbol{x}(t, \boldsymbol{z})) \\ \quad \boldsymbol{x}(0, \boldsymbol{z}) = \boldsymbol{z}. \end{cases} \tag{4.5}$$

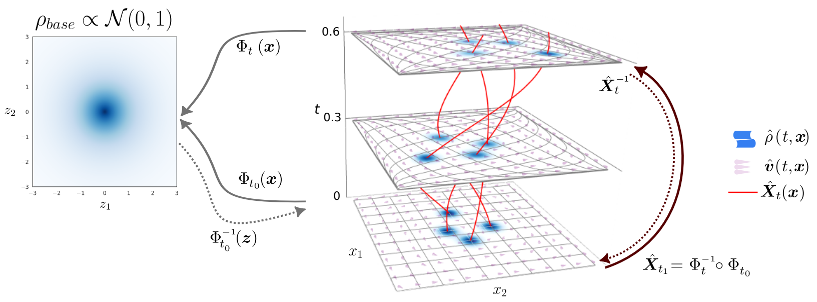
In SLDA, the density at the departure points (i. e. at the initial time) ρ_0 is represented by an interpolated mesh instead of a fixed base distribution. Similarly, the fields governing the dynamics (in our case, just the velocity) are parameterized by spatio-temporal (3D+time) meshes that are interpolated. The data-loss for the density is computed by mapping the observations backward in time with Eq. (4.4) and then comparing it with the interpolated initial density. Note that this is identical to evaluating the log-likelihood in a CNF.

4.3 LAGRANGIAN FLOW NETWORKS

We first present some key results of the classical theory of Lagrangian flows for smooth vector fields. These provide us a framework for evolving densities and velocities that always fulfill the CE. We then propose parameterizations that result in simple expressions for the velocity and density. The resulting LFlow models densities and velocities by building upon conditional Normalizing Flows.

4.3.1 *Flow Maps and the Continuity Equation*

The Lagrangian view describes fluids from the perspective of moving fluid parcels, i.e. infinitesimal volumes with constant mass. From this point of



$\rho_{base} \propto \mathcal{N}(0,1)$
z_2
z_1
$\Phi_t(\boldsymbol{x})$
$\Phi_{t_0}(\boldsymbol{x})$
$\Phi_{t_0}^{-1}(\boldsymbol{z})$
t
0.6
0.3
0
x_1
x_2
$\hat{\boldsymbol{X}}_t^{-1}$
$\hat{\boldsymbol{X}}_{t_1} = \Phi_t^{-1} \circ \Phi_{t_0}$
$\hat{\rho}(t,\boldsymbol{x})$
$\hat{\boldsymbol{v}}(t,\boldsymbol{x})$
$\hat{\boldsymbol{X}}_t(\boldsymbol{x})$

FIGURE 4.1: Illustration of the transformations and involved fields for modeling the temporal evolution of a 2D density with LFlows. The red lines inbetween the planes indicate trajectories of fluid parcels.

view the CE states that density changes of the fluid are described by volume changes of parcels. That is, spatial contraction increases the density of a parcel, and expansion decreases it. In order to compute the density of any parcel, we then only need to know its initial density, and how much its volume was distorted.

More formally, let $\boldsymbol{x}_{t_0}$ denote the initial position of a parcel at time t_0. In addition, let $\boldsymbol{X}_t : \Omega \mapsto \Omega$ for a fixed $t \in [t_0, T]$ be a diffeomorphism that maps $\boldsymbol{x}_{t_0}$ to the parcel position at time t:

$$\boldsymbol{X}_t(\boldsymbol{x}_{t_0}) = \boldsymbol{x}_t, \tag{4.6}$$

with $\boldsymbol{X}_{t_0}$ being the identity map. That is, $\boldsymbol{X}_t$ provides the continuous trajectory of the parcel $\boldsymbol{x}_{t_0}$. We further assume basic regularity of $\boldsymbol{X}_t$ and $\boldsymbol{X}_t^{-1}$ such as smoothness and globally bounded derivatives. Since $\boldsymbol{X}_t$ provides the trajectory of a parcel, the velocity of a given parcel at position $\boldsymbol{x}_t$ and time t follows naturally. First, the parcel is mapped back to its initial position with $\boldsymbol{X}_t^{-1}$. The velocity is then the change in position along its trajectory with respect to time t:

$$\boldsymbol{v}(t,\boldsymbol{x}) = \frac{\partial \boldsymbol{X}_t}{\partial t}\left(\boldsymbol{X}_t^{-1}(\boldsymbol{x})\right). \tag{4.7}$$

Following classical Cauchy Lipschitz theory, it is known that such a map $\boldsymbol{X}_t$ is the unique flow map of $\boldsymbol{v}$ starting at time t_0. Specifically, for any $\boldsymbol{x} \in \Omega$ the curve $t \mapsto \boldsymbol{X}_t(\boldsymbol{x})$ is the unique solution to the Cauchy Problem

$$\begin{cases} \partial_t \boldsymbol{X}_t\left(\boldsymbol{x}\right) = \boldsymbol{v}\left(t, \boldsymbol{X}_t(\boldsymbol{x})\right) & t \in [t_0, T), \\ \boldsymbol{X}_{t_0}(\boldsymbol{x}) = \boldsymbol{x}. \end{cases} \tag{4.8}$$

A more complete statement is given in the Appendix B.1.1 Theorem 4, and we refer to Hartman (2002) for an extensive description of the theory of ordinary differential equations.

With the velocity given by Eq. (4.7), we further need to define a density to describe a fluid. Let $\rho_{t_0} : \Omega \mapsto \mathbb{R}_+$ be the (known) initial fluid density at time t_0. We can then define the time-evolved density as a transformation of ρ_{t_0} using the change of variables formula:

$$\rho(t, \boldsymbol{x}) = \rho_{t_0}\left(\boldsymbol{X}_t^{-1}(\boldsymbol{x})\right)|\det J\boldsymbol{X}_t^{-1}(\boldsymbol{x})|. \tag{4.9}$$

Given the velocity in Eq. (4.7) and a smooth ρ_{t_0}, Eq. (4.9) is a solution to the continuity equation (see B.1.1 Theorem 5).

Proofs for this statement vary significantly in their complexity and depend on the regularity assumptions for the velocity. That is, they range from classical theory to current mathematical research. For the sake of completeness, we provide precise statements, assumptions and proofs in the Appendix Section B.1. The appended proofs are based on the well-established classical theory for the existence and uniqueness of Lagrangian flows for smooth vector fields and we refer to Ambrosio and Crippa (2008) for basic and advanced results.

4.3.2 *Lagrangian Flow Networks*

We can now exploit the derived connection between the CE and time-evolving diffeomorphisms to model densities and velocities that satisfy the CE by construction. Instead of directly parameterizing both $\boldsymbol{X}_t$ and ρ_{t_0} (as done in the concurrent work of L. Li et al. (2023)), we model the density at each time ρ_t, including ρ_{t_0}, as a transformation of a simple fixed density ρ_{base}. This requires only a single time-conditioned bijection Φ_t. We call the resulting model Lagrangian Flow Networks (LFlows) and provide a high-level illustration in Figure 4.1.

Let $\Phi_t : \Omega \mapsto \mathbb{R}^d$ be a learnable diffeomorphism with $t \in [t_0, T]$. We propose to parameterize $\boldsymbol{X}_t$ as the composition

$$\hat{\boldsymbol{X}}_t(\boldsymbol{x}) = \Phi_t^{-1}\left(\Phi_{t_0}(\boldsymbol{x})\right). \tag{4.10}$$

In practice, we implement Φ_t as an invertible neural network with its parameters conditioned on time. We further define the initial density as a transformation of a simple base density:

$$\hat{\rho}_{t_0}(\boldsymbol{x}) = \rho_{\text{base}}\left(\Phi_{t_0}(\boldsymbol{x})\right)\,\left|\det J\Phi_{t_0}(\boldsymbol{x})\right|. \tag{4.11}$$

The base density $\rho_{\text{base}} : \mathbb{R}^d \mapsto \mathbb{R}_+$ is an unnormalized probability density:

$$\rho_{\text{base}}(\boldsymbol{z}) = c \cdot \mathcal{N}(\mathbf{0}, I), \tag{4.12}$$

where $c \in \mathbb{R}_+$ is the total mass of the system and a freely learnable parameter. We now substitute $\boldsymbol{X}_t$ in Eq. (4.9) with the parameterized $\hat{\boldsymbol{X}}_t$ from Eq. (4.10). We also substitute the density ρ_{t_0} with the parameterized $\hat{\rho}_{t_0}$ (Eq. (4.11)). The modeled density then simplifies to

$$\hat{\rho}(t, \boldsymbol{x}) = \hat{\rho}_{t_0}\left(\hat{\boldsymbol{X}}_t^{-1}(\boldsymbol{x})\right) \ |\det J\hat{\boldsymbol{X}}_t^{-1}(\boldsymbol{x})| = \rho_{\text{base}}\Big(\Phi_t(\boldsymbol{x})\Big) \ \big|\det J\Phi_t(\boldsymbol{x})\big|. \tag{4.13}$$

We refer to the Appendix B.2.1 for the intermediate steps. Note that the resulting expression coincides with the change of variable formula for probability densities in Eq. (2.34). This allows us to elegantly model the evolving density through a conditional Normalizing Flow with unnormalized base density ρ_{base} and bijective layers conditioned on time Φ_t.

The parameterization of $\boldsymbol{X}_t$ with $\hat{\boldsymbol{X}}_t$ in Eq. (4.7) also results in a simple expression for the velocity:

$$\hat{\boldsymbol{v}}(t, \boldsymbol{x}) = \frac{\partial \hat{\boldsymbol{X}}_t}{\partial t}\left(\hat{\boldsymbol{X}}_t^{-1}(\boldsymbol{x})\right) = -\Big(J\Phi_t(\boldsymbol{x})\Big)^{-1} \frac{\partial \Phi_t}{\partial t}(\boldsymbol{x}). \tag{4.14}$$

We provide the explicit steps for arriving at Eq. (4.14) in the Appendix B.2.2. Note that in order to evaluate $\hat{\rho}$ and $\hat{\boldsymbol{v}}$ we now require only the forward map Φ_t, but not its inverse. This proves useful for layers with expensive inverses, or if the inverse is unknown. An illustration that unifies the Lagrangian view and the provided parametrization of LFlows is given in Figure 4.1.

4.4 IMPLEMENTATION

To implement LFlow as outlined in Section 4.3, we require conditional bijective layers. That is, the diffeomorphism Φ_t required for Eq. (4.13) and Eq. (4.14) has to be conditioned on time t. To allow for a flexible parameterization, we first embed t into a higher-dimensional space with an embedding network $f_\Theta : [t_0, T] \mapsto \mathbb{R}^k$. The k-dimensional embedding $\boldsymbol{c} = f_\Theta(t)$ is shared between the individual layers, and corresponds to the conditional input $\boldsymbol{c}$ discussed in the background Section 2.4.3. That is, individual layers consist of neural networks that take as input the time embedding $f_\Theta(t)$ and output the parameters of a diffeomorphism, resulting in time-conditioned bijections $\Phi_t := \Phi(\boldsymbol{x}; f_\Theta(t))$. A high-level visualization

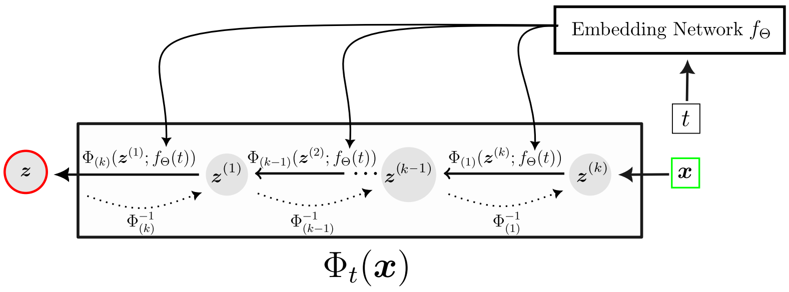


FIGURE 4.2: General Architecture of the conditional Flows. The dotted lines ($\cdots$) indicate the inverse direction). A shared embedding is created based on the time condition, and layer-wise neural networks then parameterize the bijections.

of the network architecture is provided in the Figure 4.2[1].

We implement the embedding network $f_\Theta(t)$ as MLPs with residual skip connections and swish activations (Elfwing et al., 2018). The specific composition of the bijections $\Phi(x; f_\Theta(t))$ varies depending on the experiment. However, the main building blocks are always Lipschitz-constrained invertible densenets (i-DenseNet) (Perugachi-Diaz et al., 2021). Specifically, we use a conditional variant of i-DenseNets which we introduced in the background Section 2.4.3.2 as $\mathcal{T}_{\text{RES}}(x, c)$. Further, we rely sinusoidal activations (CSin) instead of the ClipSwish activations proposed by Perugachi-Diaz et al. (2021), which we will discuss in Section 4.4.2. In addition to conditional i-DenseNet blocks, we employ (unconditional) intermediate activation normalizations (Kingma & Dhariwal, 2018) and (conditional) SVD layers, i. e. $\mathcal{T}_{\text{LN}}(x)$ and $\mathcal{T}_{\text{SVD}}(x, c)$ from the background Section 2.4. The Jacobian determinant of the i-DenseNet blocks is computed using brute force, which is still feasible in ≤ 3 dimensions.

Code to our experiments is provided in the supplementary material[2], and on Github[3].

Experiment Specific Implementations. For implementation and architecture details of LFlows and competing methods for all experiments, we refer to the Appendix Section B.3. Competing models include semi-

1 This hyper-network architecture for conditional flows is based on the code provided by the *nflows* library (Durkan et al., 2019) and not a contribution from our side.
2 https://openreview.net/attachment?id=Nshk5YpdWE&name=supplementary_material
3 https://github.com/bmda-unibas/LagrangianFlowNetworks

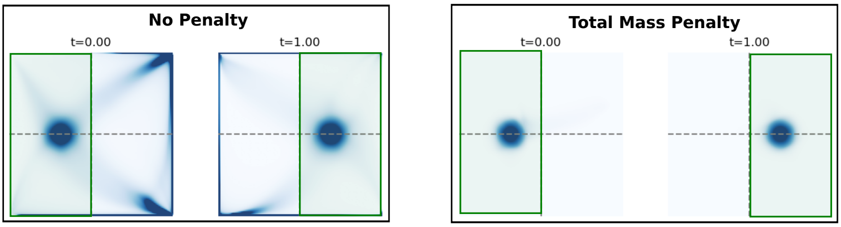


FIGURE 4.3: Predictions of the LFlow trained on two timepoints without *(left)* and with (*right*) total mass regularization. The green box indicates training points, the remainder is unobserved.

Lagrangian data assimilation (SLDA), divergence-free neural networks (DFNNs) (Richter-Powell et al., 2022), and PINNs (Raissi et al., 2019).

4.4.1 *Normalization Constant.*

In all our settings the total mass, i.e. the normalization constant c in Eq. (4.12), is not known. Therefore, we treat c as a freely learnable hyperparameter, which we initialize based on a validation set. We further encourage solutions with a small total mass by introducing a penalty on the total mass, i.e. on the learned normalization constant c:

$$L_{\text{mass}} = w_c \int_\Omega \rho_\theta(t, \boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = w_c \cdot c, \tag{4.15}$$

with the hyperparameter $w_c \in \mathbb{R}_{\geq 0}$ weighting the penalty. In practice, this penalty discourages learning significant densities in areas where there are no measurements.

To illustrate the effect of total mass regularization, we train an LFlow on a density that is only observed in a subregion of the domain. In addition, the normalization constant of the LFlow is initialized with a high value. The results are shown in Figure 4.3. The green areas indicate observed regions, i. e. the density is observed at two separate time points, once on the left and once on the right half of the domain. Without any regularization, the network will simply push the mass away from the observed region, leading to large densities directly outside of the region. The penalty on the normalization constant instead encourages the network to decrease the total mass. This leads to an effect that is similar to a zero prior on the predicted density.

4.4.2 *Sinusoidal Activations for i-DenseNets*

As mentioned in the preceding sections, we make use of i-DenseNets, a variation of invertible residual networks, for implementing the conditional bijections. Instead of relying on the base i-DenseNet, we, however, further improve the flexibility of i-DenseNets for low-dimensional inputs. We do so simply by exchanging the activation function. Specifically, we use sinusoidal activation functions motivated by SIRENs (Sitzmann et al., 2020) to enable more flexible Normalizing Flows in low-dimensional settings. Each activation h is then given by the elementwise function

$$\sigma_{\text{sin}}(x) = \sin(\omega x)/\omega, \tag{4.16}$$

with $\omega \in \mathbb{R}_{>0}$. Dividing by ω ensures $\text{Lip}(\sigma_{\text{sin}}) = 1$, as $\text{Lip}(f) = \sup |f'(x)|$ for a differentiable function f. Analogously to the CLipSwish activation, a concatenated (and still element-wise) version of these activations is given by

$$\sigma_{\text{csin}} = \begin{bmatrix} \sigma_{\text{sin}}(x)/\sqrt{2} \\ \sigma_{\text{sin}}(-x)/\sqrt{2} \end{bmatrix}. \tag{4.17}$$

Dividing by $\sqrt{2}$ again ensures a Lipschitz constant of 1 for the concatenation, following Section 2.4.1.2 Eq. (2.59). We numerically validated that, when using the CSin activations, the probability density integrates to one, and that the fix point iteration for the inverse converges. Figure 4.5 and Fig. 4.4 show a qualitative comparison of CSin activations with varying ω and the default CLipSwish activation (Perugachi-Diaz et al., 2021). For a more extensive evaluation we refer to Section 4.6.

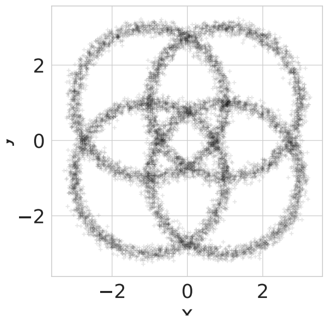


FIGURE 4.4: 10 000 Groundtruth samples from the *Four Circles* toy dataset.

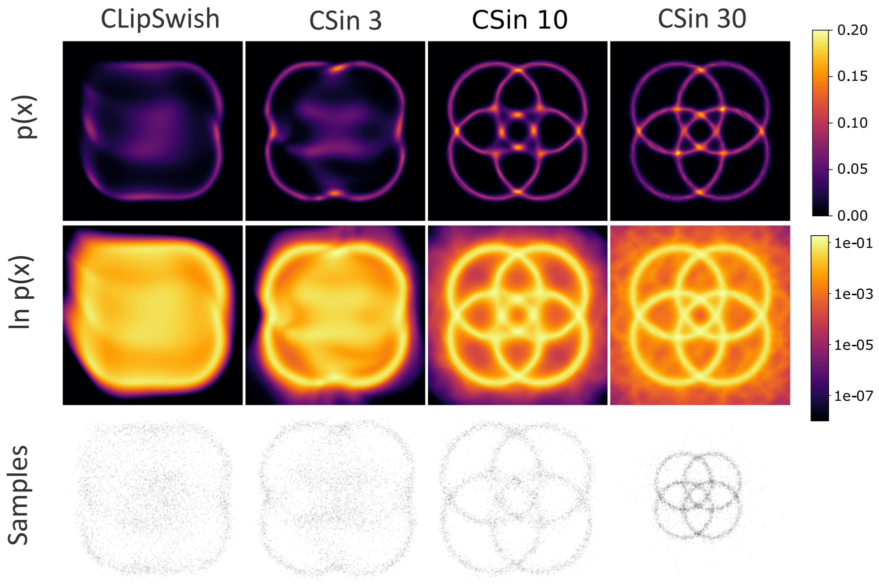


FIGURE 4.5: Learned probability densities and samples for the *Four Circles* toy dataset. Two identical i-DenseNets with different activations are trained on 2D densities with maximum likelihood. The numbers of the CSin activations indicate the factor ω. The scatter plot axes for the samples are adjusted based on the samples.

4.5 EXPERIMENTS: LAGRANGIAN FLOW NETWORKS

We showcase LFlows for two distinct settings. In setting (i), density and velocity are observed, but no equations are available (Section 4.5.1 and 4.5.3). In setting (ii) only the density is observed but further equations for the velocity are known (Section 4.5.2). For details on the data and architecture for each experiment, we refer to the Appendix B.4.1 to B.4.3. We will compare LFlows with methods that enforce the CE through different means, namely divergence-free neural networks (DFNNs), physics-informed neural networks (PINNs), and semi-Lagrangian data assimilation (SLDA). For details on the implementation we refer to the Appendix B.3.

Numerical Evaluation of Physical Consistency. Physical consistency in terms of the CE implies that the predicted density at any time coincides with the inital density transformed forward in time by the learnt velocity field. An inconsistent model would imply that the predicted velocity field fundamentally disagrees with the predicted density movements. This would make any downstream interpretation of the two learned fields futile. We quantitatively evaluate this potential inconsistency in the following experiments. We compare the predicted density $\hat{\rho}_{\text{model}}(t, \boldsymbol{x})$ with the numerical solution of the IVP defined by $\hat{\rho}_{\text{model}}(t_0, \boldsymbol{x})$, $\hat{\boldsymbol{v}}_{\text{model}}(t, \boldsymbol{x})$, and the continuity equation, i. e. Definition 2. We evaluate the symmetric mean absolute percentage error

$$\text{sMAPE} = \frac{1}{n} \sum_{i=1}^{n} \frac{|\hat{\rho}_{\text{model}}(t_i, \boldsymbol{x}_i) - \hat{\rho}_{\text{ODE}}(t_i, \boldsymbol{x}_i)|}{|\hat{\rho}_{\text{model}}(t_i, \boldsymbol{x}_i)| + |\hat{\rho}_{\text{ODE}}(t_i, \boldsymbol{x}_i)|}, \quad 0 \leq \text{sMAPE} \leq 1. \quad (4.18)$$

4.5.1 *Simulated Densities*

For a synthetic example of setting (i) we simulate densities in 2D and 3D over time by transforming a mixture of four unnormalized Gaussians.

Experimental Setup. We parameterize time-dependent bijections in $t \in [0, 1.2], \Omega = (-4, 4)^d$, which provide us analytical forms for the densities and velocities. During training, only sub-regions of the domain Ω are observed. The dynamics in 3D are similar to the 2D setting, with the xy-velocity being the same for all z values and the z velocity being 0. The only added difficulty is a higher-dimensional domain. We limit all models to the computing resources of a NVIDIA Titan X Pascal, and optimize based on the explained variance[4] (R^2) of the density on the validation set. For the PINN this resulted in 2^{16} collocation points. We do not include consistency results for DFNNs due to numerical instabilities. As DFNNs only provide access to the flux $\boldsymbol{F}$, the velocity $\boldsymbol{v} = \boldsymbol{F}/\rho$ becomes numerically unstable in low-density regions, which are abundant in this experiment.

Results. Results for 10 random seeds are provided in Figure 4.7. In addition, snapshots of the 3D prediction for $z = 0$ are shown. All methods aside from the PINN have a low consistency sMAPE on the order of 1e-4 or lower. This is expected, as LFlows enforce the CE by construction. Furthermore, SLDA computes the density similarly to our numerical reference, although with a lower order ODE solver. Low error tolerances or low-order solvers for SLDA would of course still result in inconsistencies. Looking at the predictive performance, LFlows show the highest average R^2 for the density in both 2D and 3D. While PINNs perform competitively in 2D, they severely degrade in 3D, as the number of collocation points used (limited by GPU memory) is insufficient to enforce the PDE. This is also reflected in their increase of the consistency sMAPE in 3D. Finally, DFNNs and SLDA are roughly comparable in terms of predictive accuracy, with DFNNs being in our experience most prone to overfitting. We note that small displacements of the density can already lead to large differences in R^2.

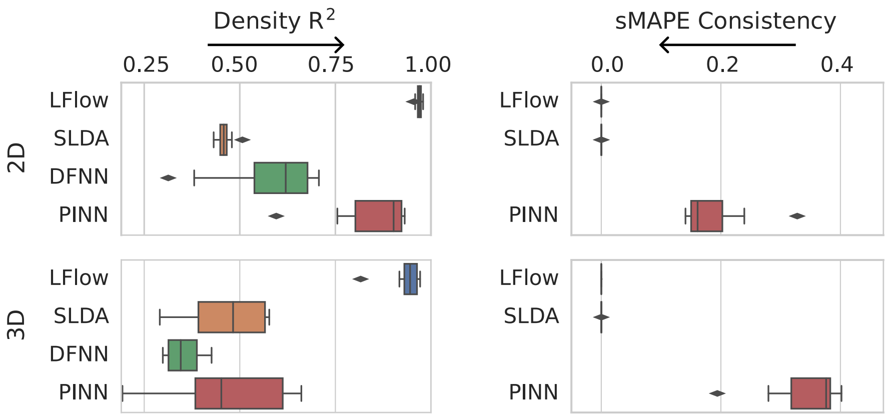


FIGURE 4.6: Results on the synthetic example in 2D and 3D. Left: Test R2 for the predicted density (higher is better). Right: Consistency loss (lower is better). Missing for DFNN due to numerical issues.

4 $R^2 = 1 - \frac{MSE(y_{obs},\hat{y})}{Var(y_{obs})} \leq 1$ with $R^2 = 1$ indicating a perfect reconstruction.

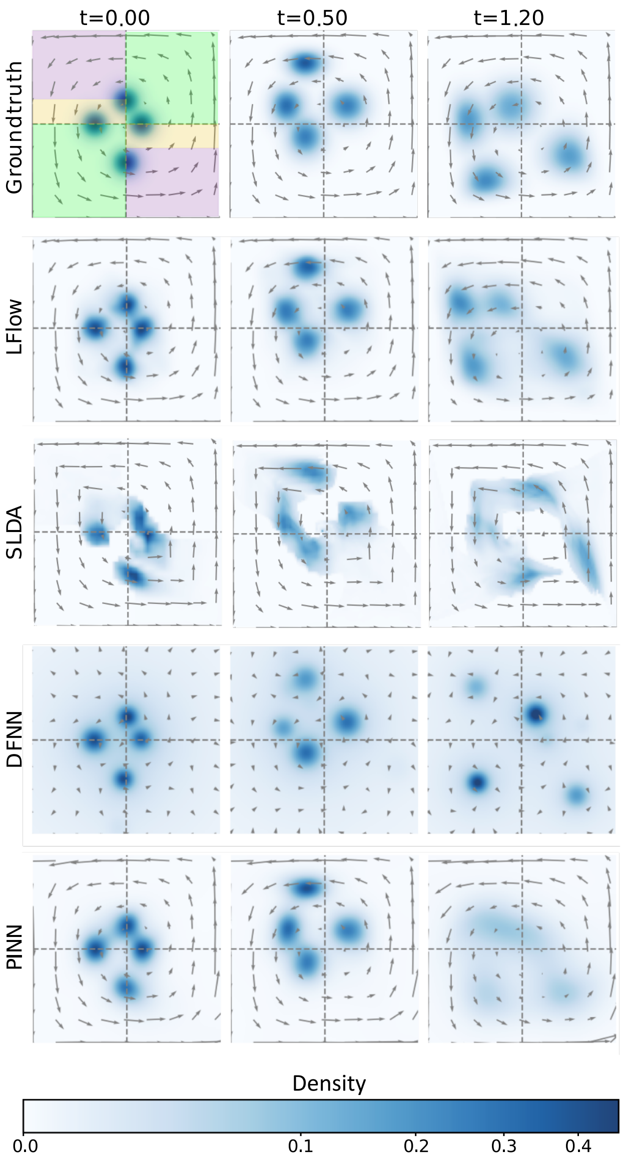


FIGURE 4.7: Evaluation of the predicted density on the xy-plane with $z = 0$ for different methods. Arrows indicate velocity, except for the DFNNs, where the normalized flux is shown. The lower left plot shows the spatial splitting into train (green), validation (yellow), and test (purple) subsets.

4.5.2 *Dynamical Optimal Transport*

As an example of setting (ii), in which no velocity is observed but additional equations dictating the dynamics are known, we consider dynamical optimal transport problems. Our experimental setting closely follows Richter-Powell et al. (2022). Specifically, we consider the Benamou-Brenier formulation of the optimal transport problem between two densities p_{t_0} and p_{t_1} (Benamou & Brenier, 2000). In this case the optimal transport map is the solution map $\boldsymbol{X}_t$ of a flow that is defined by the vector field $\boldsymbol{v}$ and minimizes the following objective:

$$\min_{\boldsymbol{v},\rho} \int_{t_0}^{t_1} \int_{\Omega} |\boldsymbol{v}(t,\boldsymbol{x})|^2 \rho(t,\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t \tag{4.19}$$

subject to the constraints $\rho(t_0,\boldsymbol{x}) = p_{t_0}(\boldsymbol{x})$ and $\rho(t_1,\boldsymbol{x}) = p_{t_1}(\boldsymbol{x})$. Furthermore, ρ and $\boldsymbol{v}$ are subject to the continuity equation $\partial_t \rho = -\nabla \cdot (\rho \boldsymbol{v})$.

Both DFNNs and LFlows can solve the minimization problem in Eq. (4.19) without needing a separate estimation of ρ. Instead, one fits the densities at t_0 and t_1 and additionally minimize Eq. (4.19). However, to obtain the transport map from the learned velocity field, DFNNs need to numerically solve the Cauchy problem in Eq. (4.8). An ODE solver might however struggle due to the numerical instability of the DFNNs velocity in low-density regions. In contrast, LFlows elegantly provide an analytical form for the continuous transport map through the learned bijections, i.e. $\hat{\boldsymbol{X}}_t(\boldsymbol{x}) = \Phi_t^{-1}(\Phi_{t_0}(\boldsymbol{x}))$.

Experimental Setup. We train the models by minimizing the loss

$$L(\hat{\rho},\hat{\boldsymbol{v}}) = L_{\text{obs}}(\hat{\rho},\hat{\boldsymbol{v}}) + L_{\text{ot}}(\hat{\rho},\hat{\boldsymbol{v}}) \tag{4.20}$$

$$L_{\text{obs}}(\hat{\rho},\hat{\boldsymbol{v}}) = \lambda \mathbb{E}_{\tilde{p}_0}\left[|\hat{\rho}(0,\boldsymbol{x}) - p_0(\boldsymbol{x})|\right] + \lambda \mathbb{E}_{\tilde{p}_1}\left[|\hat{\rho}(1,\boldsymbol{x}) - p_1(\boldsymbol{x})|\right] \tag{4.21}$$

$$L_{\text{ot}}(\hat{\rho},\hat{\boldsymbol{v}}) = \int_0^1 \int_{\Omega} |\hat{\boldsymbol{v}}(t,\boldsymbol{x})|^2 \hat{\rho}(t,\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}t \tag{4.22}$$

where data is drawn from $\tilde{p}_i$, which is a mixture of p_i and a uniform density (for $i = 0,1$); λ is a hyperparameter. At test time we empirically estimate the W_2^2 distance by mapping 5000 samples of p_0 from $t = 0$ to $t = 1$. Different to Richter-Powell et al. (2022) we repeat this estimate 50 times. We compare the W_2^2 estimates of (i) LFlows, (ii) DFNNs and (iii) a minimax formulation

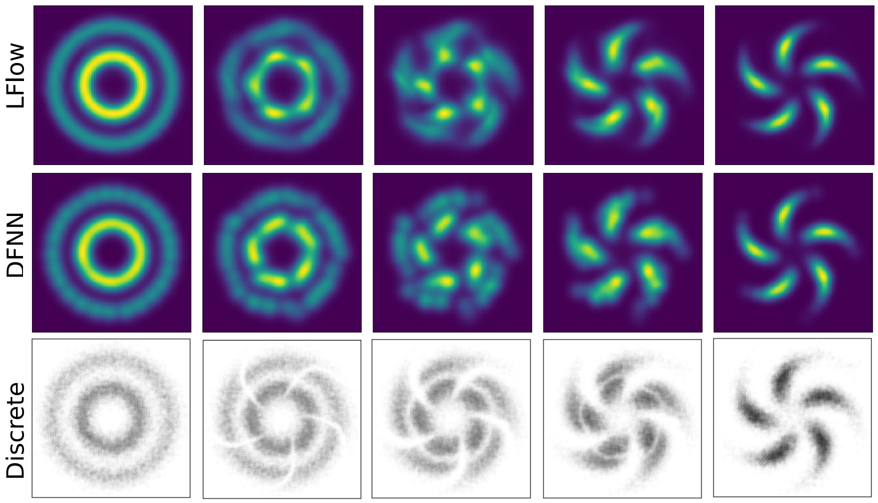


FIGURE 4.8: Approximations of the 2D optimal transport map with LFlows, DFNN and a discrete reference. Dataset: Circles $\rightarrow$ Pinwheel.

of the optimal transport map learned via input convex neural network (Makkuva et al., 2020). For DFNNs the samples are transported through the estimated velocity with an ODE solver. We further compare to the W_2^2 estimates of a discrete OT-solver (Bonneel et al., 2011) from the *pot* library (Flamary et al., 2021) based on 50000 samples. The λ for LFlows is chosen by matching the W_2^2 distance between the *moons* and *swissroll* datasets with the discrete estimate. For DFNNs a λ value is provided by Richter-Powell et al. (2022). In this experiment we restrict ourselves to methods that can exactly enforce the CE and thus exclude PINNs.

Results. We considered three different pairs of toy 2D distributions. Figure 4.8 shows the approximated optimal transport maps learned with LFlows and the W_2^2 estimates of the different methods. We verified that the DFNN and the LFlow fit the target densities well, with a test MSEs below 8e-5 for all settings. We excluded SLDA, as it was unstable and did not consistently result in low errors for the two target densities. The LFlow estimates of W_2^2 are closest to the range of discrete estimates (Figure 4.9). In contrast, the minimax model underestimates the distance which is consistent with the results of Richter-Powell et al. (2022). DFNNs significantly overestimated W_2^2 and we were unable to fully reproduce results obtained

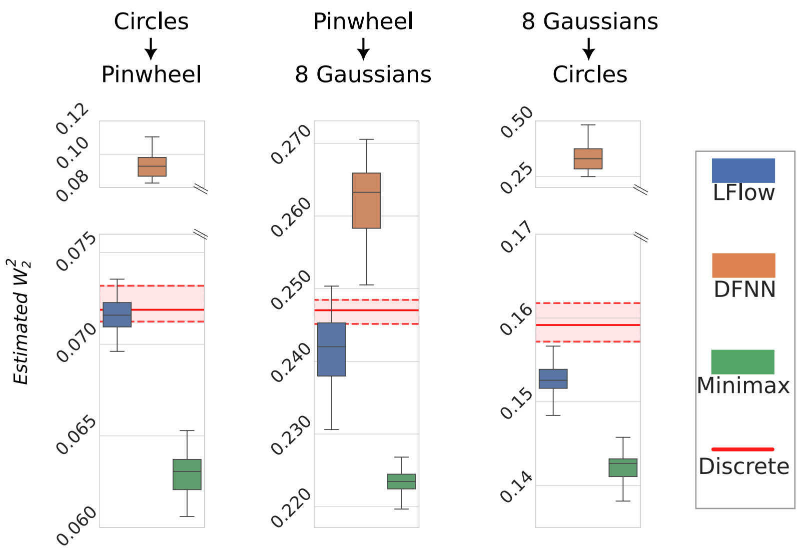


FIGURE 4.9: Estimated Wasserstein distances of the 2D optimal transport map with LFlows, DFNN and a discrete reference. The red vertical lines denote the minimum, median, and maximum estimates of 5 runs with a discrete OT solver.

by Richter-Powell et al. (2022). We assume that this is due to the ODE solver struggling with the unstable velocity calculation in low-density regions.

4.5.3 *Modeling Bird Migration*

As a real-world application for setting (i) we model bird migration within Europe based on weather radar measurements. The data provided by Nussbaumer et al. (2021) contains estimated bird densities (birds/km^3) and velocities (m/s). Measurements are taken from 37 weather radar stations in France, Germany, and the Netherlands at up to 5-minute intervals at 200m altitude bins, reaching up to 5km. The velocity data does not include a z-axis component.

Experimental Setup. We model the bird migration of 3 subsequent nights of April 2018. We assume that the mass is mostly conserved within the three nights during migration. We test the predictions on radars located in the center of the covered region, which were excluded during training (see

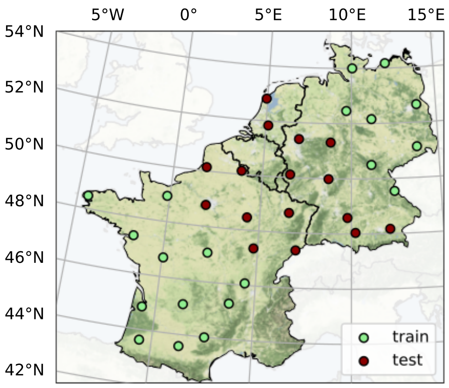


FIGURE 4.10: Locations of the weather radars within central Europe that were used to obtain bird densities for the considered data set.

Figure 4.11). Hyperparameters of all models are selected by minimizing the density MSE on three nights of March 2018. As a baseline, we compare the results with a 10-layer multilayer perceptron (MLP) with skip connections, 256 hidden units per layer, ReLU activations, and batch normalization. Additionally, we evaluate the PINN, DFNN, and SLDA. The PINN has the same general architecture as the MLP but additionally minimizes the penalty $\lambda \cdot \|\partial_t \hat{\rho} + \nabla \cdot (\hat{\rho}\hat{v})\|_2$ evaluated on 100 000 collocation points, where $\lambda \in \mathbb{R}_+$ is a hyperparameter.

Results. Figure 4.11 shows snapshots of the vertically integrated density and flux predicted by LFlows. Predictions of the other models are provided in the Appendix B.4.3.2. Aside from the velocity and density, LFlows readily provide the trajectories (shown in orange). In practice, experts could compare these with the migration paths taken by individual birds, which are for example obtained from bird-ringing studies.

We compare test density errors for each model in Figure 4.12 (a). The methods that enforce the continuity equation result in a lower error than the baseline MLP. Furthermore, the consistency sMAPE in Figure 4.12 (b) shows that PINN and MLP lead to an inconsistent density and velocity. LFlows, SLDA and DFNNs have a sMAPE that is on the order of 1e-4 or lower. Figure 4.13 visualizes the pointwise values of the sMAPE for the MLP, PINN, and LFlow at a fixed time.

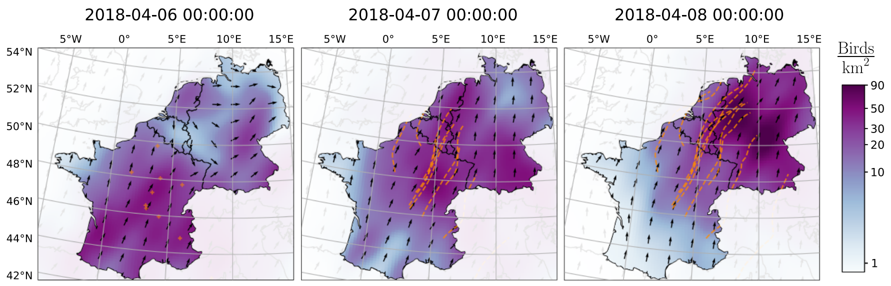


FIGURE 4.11: Snapshots of predicted bird density at three consecutive nights within central Europe. The 2D projection was obtained by integrating over altitudes covered by the radars. Orange lines indicate 2D (xy) projections of 3D trajectories from t_0 to t using randomly sampled departure points.

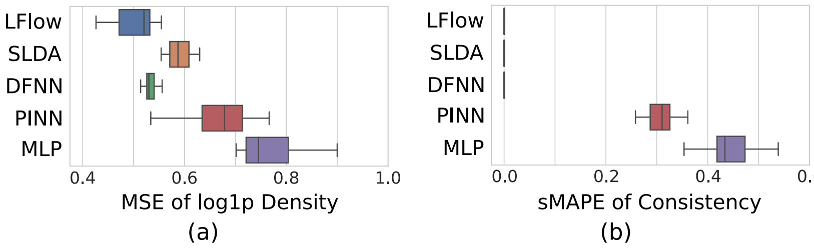


FIGURE 4.12: *(a)* Test MSE of the (log1p) density and *(b)* consistency sMAPE evaluated at multiple timesteps.

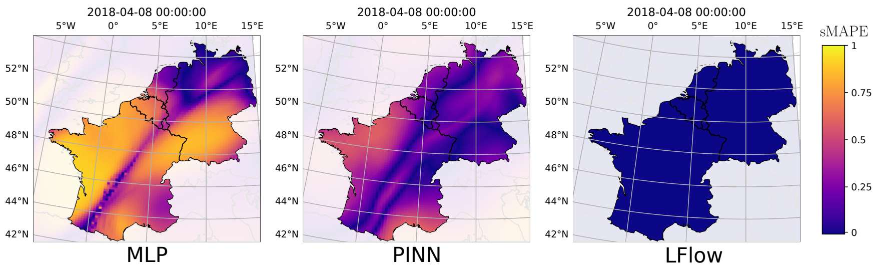


FIGURE 4.13: Consistency sMAPE of MLP, PINN and LFlow at single time point (2018-04-08 00:00).

Computational Costs. While DFNNs and SLDA are nearly competitive with LFlows in terms of the density MSE, they suffer from high computational costs. SLDA requires to evaluate a neural network many times to numerically solve the ODE. In addition, SLDA gets slower during training because the dynamics given by the neural network become more stiff. This is shown by the increase in run time for each epoch in Figure 4.14 (a) and is a known limitation of models in the neural ODE family (Biloš et al., 2021). DFNNs on the other hand result in huge memory requirements due to the required second-order derivatives. Figure 4.14 (b) shows the peak memory use in terms of VRAM for varying minibatch sizes of the models. In practice, high memory requirements result in small minibatch sizes, which ultimately lead to a slower training and inference pipeline (Shallue et al., 2018). The high peak memory and runtime of PINNs is due to the large amount of collocation points, which could be alleviated with more sophisticated sampling methods.

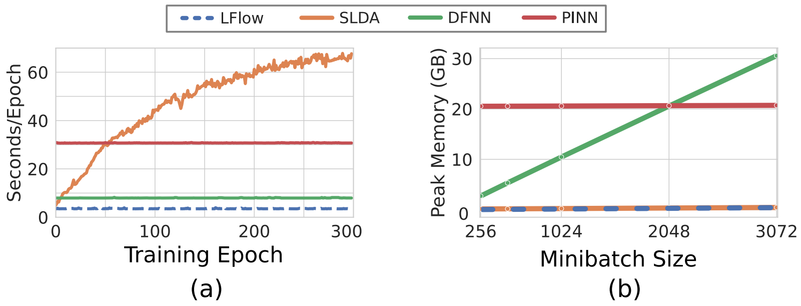


FIGURE 4.14: *(a)* Time per training epoch during training and *(b)* peak memory usage during training in GB VRAM for varying minibatch sizes for the bird experiment.

Total Mass Regularization. To illustrate the effect of the total mass penalty in a real-world setting, we train an LFlow on the bird migration problem with varying penalty weights, as shown in Fig. 4.15.

We further take a look at the variation of the density for differing penalty weights, which can be shown in a single visualization. Similarly to before, we train different models for varying total mass penalty weights. We calculate the relative standard deviation of the predicted flux (i.e. the product of density and velocity) at each spatial location for a fixed time. Areas with higher relative standard deviations (std) correspond to areas with largely varying explanations. This variation could be seen as an ad hoc measure of

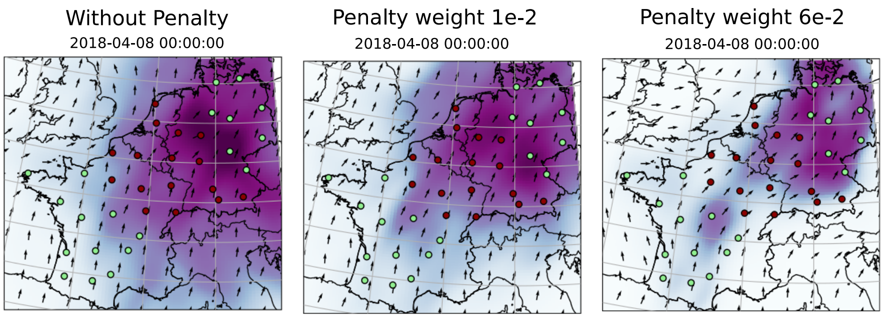


FIGURE 4.15: Predictions of the LFlow trained without *(left)* and with (*right*) total mass penalty. While predictions at observed radar stations barely change, the total mass outside of the observed region is significantly reduced.

uncertainty calculated from an ensemble of models. Figure 4.16 shows the resulting map for a single time frame for the bird migration problem. Areas closer to train radar stations (light green points) have, as expected, a lower relative standard deviation, and the areas that are never observed show the highest variation. An interesting result is, however, that the left-out central region has a relatively low deviation of predictions. That is, the variation does not seem to be a mere result of the distance to the measurements.

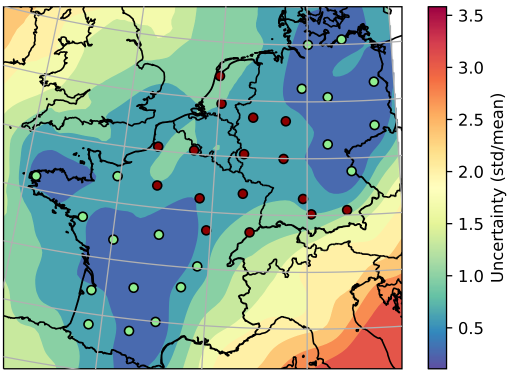
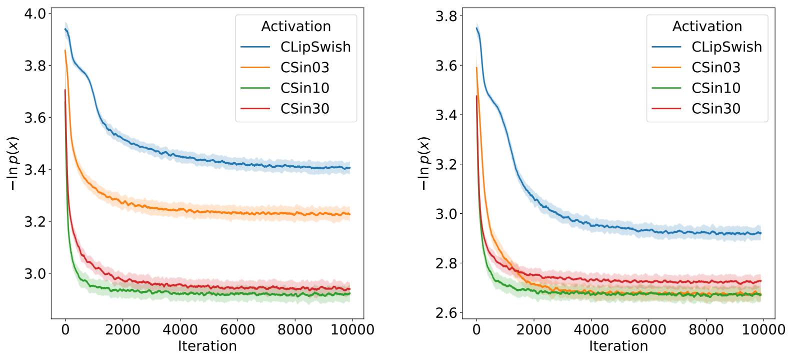


FIGURE 4.16: Relative standard deviation of the predicted (ρv) with varying mass penalties. Snapshot of a single time frame.

4.6 EXPERIMENTS: I-DENSENETS WITH CSIN

As an addendum, we evaluate i-DenseNet with CSin activations on 2D toy problems introduced in Section 4.4.2. This section solely serves as a showcase of the CSin activations, and is independent of the proposed LFlows.

Experimental Setup. For the evaluation, we consider probabilistic density estimation tasks with an unconditional Normalizing Flow. The i-DenseNets are trained via maximum likelihood on samples from the target distribution. We refer to Appendix B.4.4 for details on architecture and training.



(a) Dataset: Four Circles (b) Dataset: Two Spirals

FIGURE 4.17: Negative Log-Likelihood of the (a) *Four Circles* and (b) *Two Spirals* toy dataset for an i-DenseNet with different activation functions. The line indicates the running mean over 100 iterations and the shaded region the corresponding standard deviation.

Results. Fig. 4.5 shows qualitative result for the *Four Circles* data set using the same i-DenseNet with CLipswish or CSin activation with different choices for ω. Fig. 4.17 provides the corresponding log-likelihood during training. All CSin activations lead to higher flexibility and faster convergence during training, although high ω result in a widely spread low density in areas without observations. Longer training or varying learning rates resulted in qualitatively similar results and differences between activation functions for various toy datasets. For a more quantitative evaluation, we compare CSin ($\omega = 10$) with the CLipSwish results reported by Perugachi-Diaz et al. (2021) on 3 toy datasets over 5 repeated runs with different seeds. We specifically explore a much smaller model (≈ 5600 parameters) compared to the ones used in Perugachi-Diaz et al. (2021) (≈ 504000 parameters). Table 4.1 shows that an i-DenseNet with CSin activations is competitive or slightly outperforms the CLipSwish version while using roughly 100 times fewer parameters. When ClipSwish activations are used with the same number of parameters, CSin activations achieve higher log-likelihoods.

TABLE 4.1: Evaluation of the CSin activation on 2D toy datasets. The entries show the mean ± standard deviation of the negative log-likelihoods in nats (lower is better) for 5 runs with different seeds. The asterisk * indicates results taken from existing literature.

	ClipSwish*	ClipSwish	CSin10
(# Parameters)	(504K)	(5.6K)	(5.6K)
Two Moons	2.39	2.42 ± 0.007	2.37 ± 0.071
Two Circles	3.30	3.64 ± 0.044	**3.28** ± 0.006
Checkerboard	3.66	3.86 ± 0.015	**3.57** ± 0.003
Four Circles	-	3.44 ± 0.042	**2.92** ± 0.006

Finally, we also evaluate the i-DenseNet with CSin ($\omega = 15$) activations on a 6D density estimation dataset from the UCI dataset repository (Dheeru & Taniskidou, 2017), specifically the *Power* dataset. table 4.3 show the test log-likelihood compared to results of a few selected methods to put its performance into context. Although significantly outperforming Masked Autoregressive Flow (MAF) (Papamakarios et al., 2017) and CNF-based FFJORD (Grathwohl et al., 2019), the (to our knowledge) current state-of-the-art method using Neural Spline Flows (RQ-NSF) (Durkan et al., 2019) still performs best. We note that the hyperparameter tuning on our side was very limited, so a slight improvement is expected with further effort.

In summary, we have shown in this section that sinusoidal activations improve the flexibility of invertible residual networks in low-dimensional settings, which makes them suitable for LFlows.

Table 4.3: Test negative log-likelihoods for probabilistic density estimation on the 6D UCI power dataset (lower is better). The entries show the mean estimate ± 2 times the standard error of the mean for a single run, following the evaluation method of Durkan et al. (2019). The asterisk * indicates that the result is taken from existing literature. The dagger † indicates that the mean and standard deviation are with respect to multiple independent runs.

	UCI Power
i-DenseNet + CSin10	-0.57 ± 0.01
i-DenseNet + CLipSwish	-0.46 ± 0.01
FFJORD†*	-0.46 ± 0.01
MAF*	-0.45 ± 0.01
RQ-NSF*	$-\mathbf{0.66} \pm 0.01$

4.7 CONCLUSION

We introduced LFlows for modeling densities and velocities that adhere to the continuity equation by construction. We did so by establishing a link between time-conditioned diffeomorphisms and Lagrangian solution maps for the continuity equation. The resulting parametrization allows us to elegantly model time evolving densities with a single time-conditioned Normalizing Flow. Furthermore, we can calculate the velocity without inverting the conditional bijections, allowing the use of expressive bijective layers. We showed that LFlows can be applied to settings where we have sparse data on both density and velocity, and to settings where we have no data on velocity, but instead enforce additional equations.

In terms of density prediction LFlows outperform all competing models on both synthetic and real experiments. Different to methods like PINNs, which weakly enforce the continuity equation, LFlows always provide physically consistent predictions. In addition, LFlows avoid scaling limitations of DFNNs (peak memory usage) and neural adjoint based methods (training time). For downstream tasks, LFLows directly provide Lagrangian maps without the need for additional numerical solvers. In dynamical optimal transport settings, LFLows directly provide the analytic expression of the transport map. When modeling bird migration, the Lagrangian maps provide access to trajectories, which could be compared to migration paths obtained from different data modalities.

5 CONCLUSION

One of the big challenges in the field of physics-informed Machine Learning lies in the enforcement of physical constraints in form of PDEs in Machine Learning models. Our contribution focused on the continuity equation for the conservation of mass, which provides a fundamental link between the density changes of a fluid and its velocity. Satisfying this equation ensures physically consistent predictions in a model, and thus avoids contradictive interpretations of the velocity and density fields. To this end, the research surrounding this thesis has been dedicated to developing neural networks that adhere to the continuity equation in continuous space and time. In this final chapter, we summarize our main findings, review limitations of our proposed methods, and provide an outlook for future directions.

5.1 SUMMARY

The presented contributions consists of i) an extension to penalty-based enforcement of the continuity equation, and ii) a general framework for strict enforcement of the continuity equation.

Physical constraints via penalties. In our first contribution in Chapter 3 we make use of the penalty-based framework of physics-informed neural networks. We identify shortcomings of PINNs in settings with sparse and localized signals in larger domains or unknown boundaries. That is, standard PINN approaches struggle under these conditions due to their reliance on uniform sampling of the collocation points within a fixed region. Although alternative strategies for (re-)sampling collocation points exist, they often still rely on uniform proposals, which are inevitably unable to explore high-dimensional spaces.

To address these challenges, we introduced a variant of PINNs, called particle-density PINNs. By sampling collocation points from the particle density, pdPINNs not only adapt the location of the collocation points, but also provide an alternative loss objective. That is, the PDE loss is uniformly weighted over the particle positions, rather than being uniformly weighted over space. For obtaining samples from the unnormalized particle distribution, we rely on dynamic Monte Carlo methods. As the collocation points in the resulting pdPINNs are adaptively chosen based on the predicted density, pdPINNs can be considered truly mesh-free without relying on a Lagrangian viewpoint. Although motivated by the continuity equation, pdPINNs are generally applicable to time-dependent PDEs where the variable of interest can be viewed as a density. We evaluated pdPINNs in a series of synthetic experiments that covered applications to the continuity equation, the heat equation, and the Fokker-Planck equations. In regimes with few collocation points, pdPINNs result in lower test errors for predicted densities compared to competing methods, indicating a higher sample efficiency. This effect is more pronounced in high dimensions, and diminishes with large amounts of collocation points in low-dimensional settings, where uniform samples are capable of extensively covering the domain.

Physical constraints by construction. In our second contribution, which is covered by Chapter 4, we present a neural network architecture that adheres to the continuity equation by construction, called Lagrangian flow networks (LFlows). Instead of regularizing universal function approximators via penalties, we directly parameterize velocities and density that are always physically consistent. The proposed LFlows are based on a Lagrangian view of the continuity equation, which can essentially be split into two components. First, the trajectories and thus velocities of parcels are given by time-dependent diffeomorphisms, the so-called Lagrangian map. Second, any change in the density of a parcel is only due to a change in its volume. Given the Lagrangian map, the change in volume is available in the form of the (absolute) Jacobian determinant of the map. Consequently, the density of any parcel may be evaluated by mapping it back to its initial time, evaluating its initial density, and scaling it with the corresponding (absolute) Jacobian determinant.

The presented approach implements Lagrangian maps indirectly via a time conditioned Normalizing Flows, warping a fixed and rescaled base density to the target density. By doing so, there is no need to separately parameterize the initial density, which would, for example, be required for classical data assimilation methods. An important property of LFlows is that the inverse transform of the bijective layers is not needed to evaluate either density or velocity. This enables the use of flexible invertible layers that lack an analytical inverse, such as invertible residual networks. LFlows demonstrate high predictive accuracy and flexibility in both synthetic experiments and real-world applications. However, the outstanding property of LFlows is the ability to enforce the continuity equation without the drawbacks of current alternatives. Namely, neural ODE based methods can lead to stiff dynamics and thus increasingly slow training without careful regularization and encouragement of straight trajectories. Divergence-free neural networks instead lead to prohibitively high memory requirements due to the required higher-order derivatives for parameterizing non-negative densities. Finally, PINNs fail to enforce the continuity equation with sufficient accuracy, even with many collocation points. While extensions such as pdPINNs can alleviate these problems, the lack of any formal guarantee, combined with the additional need for tuning hyperparameters, remains.

5.2 LIMITATIONS AND FUTURE DIRECTIONS

After summarizing the main contributions and findings of our research, we now turn our attention to examining the limitations of pdPINNs and LFlows. Since limitations often suggest potential areas for improvement, we directly discuss possible avenues for future research.

5.2.1 *Particle-density PINNs*

Generalization to different settings. We proposed pdPINNs for general settings, where the variable of interest can be viewed as the physical density. In addition, we assumed that the interest lies in accurately predicting the density with a focus on enforcing the PDE in high-density regions. However, these assumptions strongly depend on the application at hand. For example, the interest may not be on the main bulk of density, but rather on the behavior of the velocity in regions of low density. With such an objective, pdPINNs on their own would not be beneficial. However, the mesh-free properties of pdPINN can still be advantageous. It should also be noted that pdPINNs are not mutually exclusive with additional residual-based refinement methods, such as RAR. Instead of relying on uniform proposals, it would be straightforward to draw proposals for such refinement methods from the particle distribution.

5.2.2 *Lagrangian Flow Networks*

Although Lagrangian flow networks provide strict enforcement of the continuity equation, the reliance on bijective layers also results in some restrictions and limitations inherent to the framework.

Expressiveness of bijective layers. LFlows model fluid densities and velocities by transforming a base density with bijective layers. As such, similar limitations as for Normalizing Flows apply. If the target density has disconnected modes, the base density must have the same number of disconnected modes due to topological constraints (Papamakarios et al., 2021). If not, the space inbetween disconnected modes will be covered by a small but non-zero density. Furthermore, LFlows are limited by the expressive power of the bijective layers. Even though state-of-the-art bijective layers are highly flexible, each layer might still be limited in terms of the number of modes that can be modeled (Liao & He, 2021).

Going beyond physical densities. We introduced LFlows as a model for hydrodynamic flow problems, with the goal of modeling physical densities. However, such an application domain could be considered rather niche and limited. To expand the possible application areas, it might be useful to change the perspective of our contribution. Instead of modeling physical densities and velocities with the help of conditional Normalizing Flows, we could view LFlows as a way to equip conditional Normalizing Flows with a velocity. For probabilistic density estimation tasks that depend on time, this would open up a range of interpretable regularization methods for conditional NFs.

For example, Tong et al. (2020) model cellular dynamics in data from single-cell RNA sequencing, focusing on interpolating a time series of distributions. The presented TrajectoryNets are based on maximum likelihood density estimation with continuous Normalizing Flows. The data samples correspond to low-dimensional (5D) embeddings of sequencing data at different time points. Borrowing concepts from fluid dynamics, TrajectoryNet additionally regularizes the velocity of the continuous NF with a dynamical optimal transport objective. LFlows would allow conditional NFs to be used in a similar manner for any time-dependent probability density estimation task, without the need for neural ODE solvers.

For a more speculative direction, it would be interesting to study the effect of e. g. optimal transport regularizations for Normalizing Flows with multiple conditioning variables. That is, Lagrangian flow networks with multiple variables that could be interpreted as "time". A dynamical optimal transport penalty would encourage straight trajectories of constant speed with respect to each conditioning variable. That is, changes in probability would be less abrupt under changing conditions. Such an approach might provide an attractive regularization objective, especially in settings with many conditioning variables and few observations.

Finally, there is the potential to go beyond currently established regularization methods, such as dynamical optimal transport, and instead look more towards methods that characterize flows in fluid dynamics. For example, the finite time Lyapunov exponent measures the divergence of trajectories with infinitesimal close starting points and is, for example, used for studying the geophysical fluid transport in oceans (Ser-Giacomi et al., 2015). Its computation is based on the Lagrangian flow map $\boldsymbol{X}_t$, and with LFlows we are in the rare position to have closed-form access to this map. Penalizing (an approximation to) the finite-time Lyapunov exponent could provide an attractive regularization objective for encouraging stable dynamics.

Enforcing additional constraints. Additional constraints and equations other than the continuity equation still have to be enforced through PDE penalties similar to PINNs. Furthermore, computing the velocity requires first-order differentiation as a trade-off for the direct density evaluation without numerical integration.

However, some additional constraints should be enforcable by construction. For an example of which we are aware, consider incompressible flows problems. Such problems require divergence-free velocities, i. e. $\nabla \cdot \boldsymbol{v}(t, \boldsymbol{x}) = 0$. The flow $\boldsymbol{X}_t$ of such a divergence-free velocity field is volume preserving i. e. $\det J\boldsymbol{X}_t = 1$, a property which is sometimes referred to as Liouville's Theorem (see e.g. Hairer et al. (2006, Chapter 06, Lemma 9.1)). With LFlows it is possible to construct such volume preserving Lagrangian maps by leveraging volume preserving layers, such as the Householder or shift transforms. Let the conditional bijection in a LFlow be given by

$$\Phi_t = \mathcal{B} \circ \mathcal{A}_t \tag{5.1}$$

$$\Phi_t^{-1} = \mathcal{A}_t^{-1} \circ \mathcal{B}^{-1} \tag{5.2}$$

with $\mathcal{A}_t$ being a time-conditioned, volume-preserving layer, and $\mathcal{B}$ any bijective layer that is independent of time. By definition, the Jacobian determinant of $\mathcal{A}_t$ is $|\det J\mathcal{A}_t| = 1$. Then the Lagrangian Map is given by

$$\boldsymbol{X}_t = \Phi_t^{-1} \circ \Phi_{t_0} = \mathcal{A}_t^{-1} \circ \left(\mathcal{B}^{-1} \circ \mathcal{B}\right) \circ \mathcal{A}_{t_0} \tag{5.3}$$

$$= \mathcal{A}_t^{-1} \circ \mathcal{A}_{t_0} \tag{5.4}$$

and consequently $|\det J\boldsymbol{X}_t| = 1$. Future work might, for example, investigate how to enforce other constraints on the velocity in a similar manner. In addition, it is probably necessary to study the representational power of such LFlows, that is, can we construct LFlows whose velocity can universally approximate any divergence-free vector field?

Sinks and Sources. Depending on the problem at hand, the introduction of mass sources (or sinks) into the system might be of interest. This would change the conservation law to additionally consider volume sources $q_\mathrm{v} : [t_0, T] \times \Omega \mapsto \mathbb{R}$:, resulting in the differential Eulerian formulation (see e.g. Hirsch (2007))

$$\partial_t \rho = -\nabla \cdot (\rho \boldsymbol{v}) + q_\mathrm{v}, \tag{5.5}$$

or in the Lagrangian formulation

$$\frac{\mathrm{d} \ln \rho}{\mathrm{d}t} = -(\nabla \cdot \boldsymbol{v}) + q_v / \rho. \tag{5.6}$$

That is, the density of a parcel no longer depends only on the distortion of volume and its initial position, but also on all the sinks and sources encountered along its trajectory. In our estimation, it is not straightforward to extend LFlows to such settings. Nontrivial sources/sinks q_v that depend both on space and time would require numerical integration over the parcel trajectory. However, we note that this limitation is in our understanding shared by similar work on mass conservative neural networks such as DFNNs (Richter-Powell et al., 2022). To overcome this restriction, two directions for LFlows might be explored in future work.

A rather crude and ad hoc solution would be to discretize time into multiple bins and allow instantaneous changes to the density between different bins. For example, for the bird migration setting, this would correspond to modeling individual nights separately. That is, one would effectively resort to multiple disconnected flows. For LFlows, this could be achieved by additionally conditioning the base density on the binned time.

A possibly more elegant, but also more speculative direction for overcoming this limitation is to augment the d-dimensional physical space with an additional dimension. Consider $d = 3$, with physical space $\boldsymbol{x} = [x, y, z]^\top$ and an augmented space $\boldsymbol{x}' = [x, y, z, w]$, the continuity equation is then given by

$$\partial_t \rho = -\nabla_{\boldsymbol{x}'} \cdot (\rho \boldsymbol{v}) = -\nabla_{\boldsymbol{x}} \cdot (\rho \boldsymbol{v}_{\boldsymbol{x}}) - \underbrace{\partial_w (\rho v_w)}_{=-q_v} . \tag{5.7}$$

The total mass will be conserved in the augmented space $\boldsymbol{x}'$, but the total mass within a slice Δw in the new dimension would be subject to the influx and outflow along the new dimension.

5.3 CLOSING REMARKS

In this thesis, we focused on providing a set of tools to enforce the continuity equation and, more broadly, physical constraints in neural networks. In doing so, we explored different perspectives on densities and the continuity equation, as well as different ways to enforce mass conservation. Especially our introduction to LFlows presents not only a novel neural architecture, but also aims to offer the Machine Learning community insight into the Lagrangian perspective of the continuity equation. Together with the promising existing work on mass conservative neural networks, this could ideally serve as a jumping-off point for future research. Indeed, as highlighted in the previous section, the remaining limitations and challenges invite further exploration and research.

A

APPENDIX - MESH-FREE EULERIAN PINNS

This chapter contains the supplementary text from Arend Torres et al. (2022) corresponding to Chapter 3. Appendix A.1 provides the algorithm used for implementing the competing methods RAR and OT-RAR, and Appendix A.2 includes additional information on the experiments.

A.1 IMPLEMENTATION OF COMPETING METHODS (RAR, OT-RAR).

For the experimental evaluation we compare with the adaptive refinement methods *RAR* (L. Lu et al., 2021) and *OT-RAR* (Tadiparthi & Bhattacharya, 2021). Both methods rely on consecutive refinements of a fixed grid in the initial proposal. The number of collocation points is steadily increased, and collocation points once added will not be removed. To allow for a fairer comparison, we adapt both methods to use a limited budget of points, and in addition we regularly resample them. This leads to slightly modified versions of the methods, that are similar in spirit. For learning the linear mapping proposed by Tadiparthi and Bhattacharya (2021), we rely on the PyOT (Flamary et al., 2021) implementation of Knott and Smith (1984). The pseudo-code for sampling a set of collocation points is given in Algorithm 3 and Algorithm 4. The required input f_θ refers to the PDE approximated by the network, as discussed in Section 3.1. For more details on the methods, we refer to the original papers.

Algorithm 3 Adapted RAR

```
Input: $f_\theta$, uniform distribution $\mathcal{U}_\mathcal{B}$,
       number of col. points $k$, previous col. points $X_{\text{prev}}$.
  $X_{\text{prop}} \leftarrow [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_k]^T$ with $\boldsymbol{x}_i$ sampled from $\mathcal{U}_\mathcal{B}$
  $X_{\text{comb}} \leftarrow \text{concat}(X_{\text{prev}}, X_{\text{prop}})$
  $X_{\text{new}} \leftarrow \text{topk}(X_{\text{comb}}, ||f_\theta(X_{\text{comb}})||_2^2, k)$
Output: $X_{\text{new}}$
```

Algorithm 4 Adapted OT-RAR

```
Input: $f_\theta$, uniform distribution $\mathcal{U}_\mathcal{B}$,
       number of col. points $k$,
       number of points for empirical distribution $j < 2k$,
       previous col. points $X_{\text{prev}}$.
  $X_{\text{prop}} \leftarrow [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_k]^T$ with $\boldsymbol{x}_i$ sampled from $\mathcal{U}_\mathcal{B}$
  $X_{\text{comb}} \leftarrow$ concat($X_{\text{prev}}$, $X_{\text{prop}}$)
  $X_{\text{target}} \leftarrow \text{topk}(X_{\text{comb}}, ||f_\theta(X_{\text{comb}})||_2^2, j)$
  $X_{\text{source}} \leftarrow [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_j]^T$ with $\boldsymbol{x}_i$ sampled from $\mathcal{U}_\mathcal{B}$
  $M_{\text{OT}} \leftarrow \text{LinOT}(X_{\text{source}}, X_{\text{target}})$
  $X_{\text{new}} \leftarrow [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_k]^T$ with $\boldsymbol{x}_i$ sampled from $\mathcal{U}_\mathcal{B}$
  $X_{\text{map}} \leftarrow M_{\text{OT}}(X_{\text{new}})$
Output: $X_{\text{map}}$
```

A.2 ADDITIONAL EXPERIMENT DETAILS

A.2.1 *Experiments: Continuity Equation*

All experiments were run on a computing cluster using *Nvidia GeForce GTX Titan X GPUs* with 12 GB VRAM. Up to 16 Titan X GPUs could be used in parallel. In most settings, training in each experiment took less than 10 minutes.

A.2.1.1 *Data Generation*

Here we provide a more detailed description of the generated data, namely the velocity field used and the method for obtaining simulated *radar measurements*.

Velocity field. The velocity field in the xy-plane was generated from a scalar potential field $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}$ and the z-component of a vector potential

$a : \mathbb{R}^2 \to \mathbb{R}$. Through the Helmholtz decomposition[1] we can construct the velocity field $v_{\text{xy}} : \mathbb{R}^2 \to \mathbb{R}^2$:

$$v_{\text{xy}}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = -\nabla\Phi + \begin{bmatrix} \delta a/\delta y \\ -\delta a/\delta x \end{bmatrix}. \tag{A.1}$$

For both experiments, the following fields were used:

$$\Phi\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = -\frac{1}{2}(x-2)\cdot(y-2), \tag{A.2}$$

$$a\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = -\frac{1}{5}\exp\left(-\left(\frac{2}{3}x\right)^2 - \left(\frac{2}{3}y\right)^2\right). \tag{A.3}$$

The derivatives were obtained using the symbolic differentiation library *SymPy* (Meurer et al., 2017). To add a non-steady component, the resulting velocity field is modulated in amplitude as a function of time $t \in [0,3]$:

$$v_{\text{xyt}}\left(t, \begin{bmatrix} x \\ y \end{bmatrix}\right) = v_{\text{xy}}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\left(\frac{3}{2}\left|\sin\left(\frac{2}{3}\pi t\right)\right| + 0.05\right). \tag{A.4}$$

The z (altitude) component of the velocity depends only on time and is given by:

$$v_z(t) = 1.6\cdot\sin\left(\frac{4}{3}\pi t\right). \tag{A.5}$$

Simulation. For the initial distribution of the fluid, the particle positions were drawn from Gaussian mixtures. For $t \in [0,3]$, these particles were simulated using the above constructed velocity field. Overall, the paths of the roughly 240000 parcels were simulated using a basic backward Euler scheme.

Measurements. The measurements at the sensors were obtained by counting the number of particles within a given radius over multiple timesteps. The density corresponds to the mass divided by the sensor area, and the velocity is an average over all the particle velocities. For the training data, additional zero-mean isotropic Gaussian noise is added to all measurements.

1 This is the 2D formulation of the Helmholtz decomposition, where the vector potential has non-zero components only along the z-axis as in $\boldsymbol{a}_{3\text{d}} = [0,0,a]^T$. The full decomposition is commonly written as $\boldsymbol{v}_{3\text{d}} = -\nabla\Phi_{3\text{d}} + \nabla\times\boldsymbol{a}_{3\text{d}}$.

In the 3D setting, data measurements of density and velocity are obtained by 13^2 sensors on the xy plane, within the region $[-3,3]^2$ at 11 equidistant time steps. In the 2D setting, the same set of sensors is used.

A.2.1.2 *Architecture and Training*

In both experiments, the neural networks for predicint the density ρ_θ and velocity v_θ consist of fully connected layers with sinusoidal activation functions, as proposed by Sitzmann et al. (2020). The number of layers and units for each setting is shown in Table 3.1. The sine frequency hyperparameter required in the SIREN architecture was tuned by hand according to the validation loss of the baseline model (i. e. without a PDE loss), leading to a sine frequency of 12 for the 2D setting and 5 for the 3D setting. We note that the proposed default value of 30 in Sitzmann et al. (2020) leads to a heavy overfitting of our relatively low-frequency data, and thus we recommend an adjustment of this hyperparameter for usage in PINNs.

For training the network, the ADAM optimizer (Kingma & Ba, 2014) with a learning rate of 8×10^{-4} (2D Setting) or 10^{-4} (3D Setting) was used. The learning rate was multiplied by a factor of 0.99 each epoch. All models were trained for 300 (3D setting) or 500 (2D setting) epochs. The 2D setting was trained using full batch gradient descent, whereas for the 3D setting we used a mini-batch size of 6931. In all experiments, we trained and evaluated on 10 different random seeds.

A.2.2 *Experiments: Heat Equation*

A.2.2.1 *Data Generation*

Overall, the dataset is composed of 1000 training points, 1971120 test points and 492780 validation points. We made sure that training points contained enough information about the initial condition, i. e., we selected a sufficient number of points around the initial source of non-zero temperature. In contrast, the validation and test points are taken uniformly in time and space.

A.2.2.2 *Architecture in Training.*

Unlike previous experiments, the architecture used is a fully connected two-layer neural network with 32 hidden units and tanh activations. We use the ADAM optimizer (Kingma & Ba, 2014) combined with an exponential learning rate scheduler that multiplies the learning rate by a factor of 0.9999 at each epoch. The initial learning rate is 10^{-4} and decreases until a minimum value of 10^{-5}. Training was terminated through early stopping, as soon as the validation R^2 did not improve for more than 3000 epochs. During the warm-up phase of the pdPINN training, the collocation points were sampled uniformly. After the warm-up 90% of the samples were drawn from the particle density distribution, which is proportional to the modeled temperature. Collocation points were resampled every 500 epochs.

A.2.3 *Experiments: Fokker-Planck Equation*

We additionally sample (5000) collocation points at the initial time step, which is the default behavior of DeepXDE.

B APPENDIX - LAGRANGIAN FLOW NETWORKS

This appendix provides additional information on Chapter 4. Appendix B.1 covers additional theoretical background and proofs on Lagrangian flows. Appendix B.2 provides the steps to obtain the equations for calculating the density and velocity of LFlows. The following section B.3 discusses implementation details of the different methods, i. e. LFlows, SLDA, DFNNs, and PINNs. Finally, Appendix B.4 provides further information on the individual experiments, ranging from data generation to training details and additional results.

B.1 THEORETICAL BACKGROUND

Remark. *This section is an excerpt from Torres et al. (2024) and thus (nearly) identical to its Appendix Section A.1.*

In this section we provide proofs and theoretical background for the method in Section 4.3. Although the underlying theory is well established and not a novel contribution from our side, we were unable to find a single source that concisely contained all required statements written in an accessible and directly citeable manner. Section B.1.1 contains the statements we rely on in Section 4.3. The proofs for these theorems are provided in Sections B.1.2 to B.1.4

B.1.1 *Time Dependent Bijections and the Continuity Equation*

Theorem 4 provides us a velocity field given time-dependent bijections. Theorem 5 links the push-forward of the density to the solution of the continuity equation defined by an initial density and the velocity given by Theorem 4.

Theorem 4. *Let $0 \leq t_0 < T$ and let $\Omega \subset \mathbb{R}^d$ be a convex open set. Let $\boldsymbol{X} : [t_0, T] \times \Omega \to \Omega$ be a family of maps such that $\boldsymbol{X}_t : \Omega \to \Omega$ is a bijection for any $t \in [t_0, T]$ and $\boldsymbol{X}_{t_0}(\boldsymbol{x}) = \boldsymbol{x}$ for any $\boldsymbol{x} \in \Omega$. Assume that $\boldsymbol{X}$ and $\boldsymbol{X}^{-1}$ are $C^\infty([t_0, T] \times \Omega; \Omega)$ with globally bounded derivatives.*

Then, the velocity field $\boldsymbol{v}(t,\boldsymbol{x}) = \frac{\partial \boldsymbol{X}_t}{\partial t}\left(\boldsymbol{X}_t^{-1}(\boldsymbol{x})\right)$ is C^∞. In particular, $\boldsymbol{v}$ satisfies the assumptions of the Cauchy–Lipschitz Theorem 6 and $\boldsymbol{X}$ is the unique flow map of $\boldsymbol{v}$ starting at time t_0. Specifically, for any $\boldsymbol{x} \in \Omega$ the curve $t \mapsto \boldsymbol{X}_t(\boldsymbol{x})$ is the unique solution to the Cauchy Problem

$$\begin{cases} \partial_t \boldsymbol{X}_t\ (\boldsymbol{x}) = \boldsymbol{v}\,(t, \boldsymbol{X}_t(\boldsymbol{x})) & t \in [t_0, T), \\ \quad \boldsymbol{X}_{t_0}(\boldsymbol{x}) = \boldsymbol{x} \end{cases} \tag{B.1}$$

Proof. See Appendix Section B.1.3.

Theorem 5. *Let $\Omega, T, t_0, \boldsymbol{X}$ be as in Theorem 4. Given an initial density $\rho_{t_0} \in L^1(\Omega)$, we define*

$$\rho(t,\boldsymbol{x}) = \rho_{t_0}(\boldsymbol{X}_t^{-1}(\boldsymbol{x}))|\det J\boldsymbol{X}_t^{-1}(\boldsymbol{x})|. \tag{B.2}$$

Then $\rho(t,\boldsymbol{x})$ is a distributional solution to the continuity equation in Eq. (4.1) according to Definition 7, i.e. the following condition is satisfied for any test function $\phi \in C_c^\infty([t_0,T) \times \Omega)$:

$$\int_{t_0}^{T} \int_{\Omega} (\partial_t \phi + \boldsymbol{v} \cdot \nabla \phi)\rho \ \mathrm{d}x \ \mathrm{d}t = -\int_{\Omega} \rho_{t_0}(\boldsymbol{x})\phi(t_0, \boldsymbol{x}) \ \mathrm{d}x. \tag{B.3}$$

Moreover, if $\rho_{t_0} \in C^\infty(\Omega)$, then $\rho \in C^\infty([t_0,T) \times \Omega)$ and ρ is a point-wise solution to the continuity equation Eq. (4.1). If we assume in addition that $\rho_{t_0}(\boldsymbol{x}) > 0$ for any $\boldsymbol{x} \in \Omega$, then the same holds for $\rho(t,\boldsymbol{x})$ for any $(t,\boldsymbol{x}) \in [t_0,T) \times \Omega$ and ρ satisfies the log-density formula of the continuity equation

$$\frac{\mathrm{d}}{\mathrm{d}t} \ln(\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))) = -\nabla \cdot \boldsymbol{v}(t, \boldsymbol{X}_t(\boldsymbol{x})). \tag{B.4}$$

Proof. See Appendix Section B.1.2 and B.1.4.

B.1.2 *The flow associated to a Lipschitz vector field*

We recall the setting of the classical Cauchy–Lipschitz Theorem. For simplicity, let $\Omega \subset \mathbb{R}^d$ be a convex open set. Given $0 \leq t_0 < T$, let $\boldsymbol{v}\colon [t_0, T] \times \Omega \to \mathbb{R}^d$ be a bounded vector field. We say that $\boldsymbol{v}$ is Lipschitz continuous in space uniformly in time if there exists a constant $L > 0$ such that

$$|\boldsymbol{v}(t,\boldsymbol{x}) - \boldsymbol{v}(t,\boldsymbol{y})| \leq L|\boldsymbol{x} - \boldsymbol{y}| \quad \forall t \in [t_0, T]\ \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega. \tag{B.5}$$

Throughout this section, we consider maps $\boldsymbol{X} : [t_0, T] \times \Omega \to \Omega$ with the following properties:

- for any $\boldsymbol{x} \in \Omega$ the map $t \mapsto \boldsymbol{X}_t(\boldsymbol{x})$ is $C^1([t_0, T])$ with uniform bounds, namely there exists a constant $M > 0$ such that

$$|\partial_t \boldsymbol{X}_t(\boldsymbol{x})| \leq M \quad \forall t \in [t_0, T] \quad \forall \boldsymbol{x} \in \Omega; \tag{B.6}$$

- for any $t \in [t_0, T]$ the map $\boldsymbol{x} \mapsto \partial_t \boldsymbol{X}_t(\boldsymbol{x})$ is Lipschitz with uniform bounds, namely there exists a constant $L > 0$ such that

$$|\partial_t \boldsymbol{X}_t(\boldsymbol{x}) - \partial_t \boldsymbol{X}_t(\boldsymbol{y})| \leq L|\boldsymbol{x} - \boldsymbol{y}| \quad \forall t \in [t_0, T] \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega; \tag{B.7}$$

- for any $t \in [t_0, T]$ the map $\boldsymbol{x} \mapsto \boldsymbol{X}_t(\boldsymbol{x})$ is a bilipschitz transformation of Ω uniformly in time, namely $\boldsymbol{X}_t$ is a bijection of Ω and there exists a constant $C > 0$ such that

$$C^{-1}|\boldsymbol{x} - \boldsymbol{y}| \leq |\boldsymbol{X}_t(\boldsymbol{x}) - \boldsymbol{X}_t(\boldsymbol{y})| \leq C|\boldsymbol{x} - \boldsymbol{y}| \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega \quad \forall t \in [t_0, T]. \tag{B.8}$$

We state the Cauchy–Lipschitz Theorem in the case of $\mathbb{R}^d$ and for the forward flow. We refer to Hartman, 2002 for an extensive description of the theory of ordinary differential equations, as well as any book in basic differential calculus.

Theorem 6. *Given $T > 0$ and $t_0 \in [0, T)$, let $\boldsymbol{v}\colon [t_0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ be a bounded vector field that satisfies Eq. (B.5) for some constant $L > 0$. For any $\boldsymbol{x} \in \mathbb{R}^d$ there exists a unique trajectory $t \mapsto \boldsymbol{X}_t(\boldsymbol{x})$ solving the Cauchy problem*

$$\begin{cases} \partial_t \boldsymbol{X}_t\ (\boldsymbol{x}) = \boldsymbol{v}\left(\boldsymbol{X}_t(\boldsymbol{x}), t\right) & t \in [t_0, T), \\ \boldsymbol{X}_{t_0}(\boldsymbol{x}) = \boldsymbol{x} \end{cases} \tag{B.9}$$

in the integral sense. The map $\boldsymbol{X} : [t_0, T) \times \mathbb{R}^d \to \mathbb{R}^d$ is the flow of $\boldsymbol{v}$ starting at time t_0 and it satisfies Eq. (B.6), Eq. (B.7), Eq. (B.8).

Remark 1. *Under the assumptions of Theorem 6, we point out that the maps $\boldsymbol{X}_t, \boldsymbol{X}_t^{-1}$ are Lipschitz continuous for any time. Thus, given a time slice $t \in [t_0, T)$, we infer that $\boldsymbol{X}_t, \boldsymbol{X}_t^{-1}$ are differentiable almost everywhere in $\mathbb{R}^d$. Hence, the Jacobian matrices $J\boldsymbol{X}_t(\boldsymbol{x}), J\boldsymbol{X}_t^{-1}(\boldsymbol{x})$ are well defined for almost every $\boldsymbol{x} \in \mathbb{R}^d$ and we have that*

$$J\boldsymbol{X}_t(\boldsymbol{X}_t^{-1}(\boldsymbol{x})) = [J\boldsymbol{X}_t^{-1}(\boldsymbol{x})]^{-1} \quad \textit{for almost every } \boldsymbol{x} \in \mathbb{R}^d.$$

Moreover, there exists a constant $M > 0$ such that

$$|J\boldsymbol{X}_t(\boldsymbol{x})| + |J\boldsymbol{X}_t^{-1}(\boldsymbol{x})| \leq M \quad \textit{for almost every } \boldsymbol{x} \in \mathbb{R}^d.$$

Here, $|\cdot|$ is a given matrix norm (recall that all norms are equivalent in finite dimensional vector spaces). We point out that the uniqueness part of Theorem 6 and the regularity properties of the flow, as well as the existence of the Jacobian matrix, extend to Lipschitz vector field defined on a general convex domain Ω, as soon as we assume that the trajectories do not touch the boundary of Ω. In this case, we get that the flow map is a bilipschitz transformation of Ω for any time slice.

B.1.3 *Velocities from 1-Parameter Groups of Diffeomorphism*

Theorem 4 is a particular case of the following more general result. For simplicity, we consider convex domains.

Theorem 7. *Let $0 \leq t_0 < T$, let $\Omega \subset \mathbb{R}^d$ be a convex open set and let $\boldsymbol{X} : [t_0, T] \times \Omega \to \Omega$ be a family of maps on Ω such that $\boldsymbol{X}_{t_0}(\boldsymbol{x}) = \boldsymbol{x}$ for any $\boldsymbol{x} \in \Omega$. Assume that $\boldsymbol{X}$ satisfies Eq. (B.6), Eq. (B.7) and Eq. (B.8). Then, the velocity field $\boldsymbol{v}(t, \boldsymbol{x}) = \frac{\partial \boldsymbol{X}_t}{\partial t}(\boldsymbol{X}_t^{-1}(\boldsymbol{x}))$ satifies the assumptions of the Cauchy–Lipschitz Theorem 6 and $\boldsymbol{X}$ is the unique flow map of $\boldsymbol{v}$ starting at time t_0. Specifically, for any $\boldsymbol{x} \in \Omega$ the curve $t \mapsto \boldsymbol{X}_t(\boldsymbol{x})$ is the unique solution to the Cauchy problem Eq. (B.1).*

The reader might note that Theorem 7 is the inverse of Theorem 6. Indeed, given a map $\boldsymbol{X}$ satisfying all the properties of a flow map, it is natural to ask whether we can find a velocity field $\boldsymbol{v}$ within the Cauchy–Lipschitz framework whose flow starting at time t_0 is $\boldsymbol{X}$.

Proof of Theorem 4. Given $(t, \boldsymbol{x}) \in [t_0, T) \times \Omega$, we define

$$\boldsymbol{v}(t, \boldsymbol{x}) = \lim_{h \to 0} \frac{\boldsymbol{X}_{t+h}(\boldsymbol{X}_t^{-1}(\boldsymbol{x})) - \boldsymbol{X}_t(\boldsymbol{X}_t^{-1}(\boldsymbol{x}))}{h} = \partial_t \boldsymbol{X}_t(\boldsymbol{X}_t^{-1}(\boldsymbol{x})). \tag{B.10}$$

Since the map $s \mapsto \boldsymbol{X}_s(\boldsymbol{X}_t^{-1}(\boldsymbol{x}))$ is $C^1([t_0, T])$ by assumption for any $\boldsymbol{x} \in \Omega$, the velocity field $\boldsymbol{v}$ is well defined. Moreover, by Eq. (B.10), the curve $t \mapsto \boldsymbol{X}_t(\boldsymbol{x})$ solves the Cauchy problem Eq. (B.9) with $\boldsymbol{v}$ given by Eq. (B.10). We study the regularity of $\boldsymbol{v}$ to ensure that $\boldsymbol{X}$ is the *unique* flow associated to $\boldsymbol{v}$ starting at time t_0. To begin, we remark that $\boldsymbol{v}$ is uniformly bounded by Eq. (B.6). Moreover, since $\boldsymbol{X}_t^{-1}, \partial_t \boldsymbol{X}_t$ are Lipschitz maps for any time slice $t \in [t_0, T]$ by Eq. (B.7) and Eq. (B.8), we infer that $\boldsymbol{v}$ satisfies Eq. (B.5). Thus, $\boldsymbol{v}$ satisfies the assumptions of Theorem 6. In particular, $\boldsymbol{X}$ is the unique flow starting at time t_0 of the vector field $\boldsymbol{v}$. □

B.1.4 *The continuity equation*

Let $\Omega \subset \mathbb{R}^d$ be an open set, let $T > 0$ and $t_0 \in [0, T)$. Given a vector field $\boldsymbol{v} : [t_0, T] \times \Omega \to \mathbb{R}^d$, we shall consider the continuity equation Eq. (4.1) on $(t_0, T) \times \Omega$ with initial condition ρ_{t_0}. To deal with irregular vector fields and densities, we consider solutions to Eq. (4.1) in the sense of distributions. We refer to Ambrosio and Crippa (2008) for some basic and advanced results on the theory of continuity equation.

Definition 7. *Let $\Omega \subset \mathbb{R}^d$ be an open set and $0 \le t_0 < T$. Let $\boldsymbol{v} \in L^\infty([t_0, T] \times \Omega; \mathbb{R}^d)$ be a vector field and let $\rho_{t_0} \in L^1(\Omega)$. We say that $\rho \in L^\infty([t_0, T]; L^1(\Omega))$ is a distributional solution to Eq. (4.1) if the following condition is satisfied for any test function $\phi \in C_c^\infty([t_0, T) \times \Omega)$:*

$$\int_{t_0}^{T} \int_{\Omega} (\partial_t \phi + \boldsymbol{v} \cdot \nabla \phi) \rho \, \mathrm{d}x \, \mathrm{d}t = - \int_{\Omega} \rho_{t_0}(x) \phi(t_0, x) \, \mathrm{d}x. \tag{B.11}$$

Remark 2. *We point out that Definition 7 is well posed without differentiability assumptions on $\boldsymbol{v}, \rho$. However, if $\boldsymbol{v}, \rho$ are C^1 functions in time and space and ρ_{t_0} is a continuous function, after integrating by parts the left hand side of Eq. (B.11), for any test function $\phi \in C_c^\infty([t_0, T) \times \Omega)$ we obtain that*

$$\int_{t_0}^{T} \int_{\Omega} (\partial_t \rho + \nabla \cdot (\boldsymbol{v}\rho)) \phi \, \mathrm{d}x \, \mathrm{d}t + \int_{\Omega} \rho(t_0, x) \phi(t_0, x) \, \mathrm{d}x = \int_{\Omega} \rho_{t_0}(\boldsymbol{x}) \phi(t_0, \boldsymbol{x}) \, \mathrm{d}x.$$

Since $\partial_t \rho + \nabla \cdot (\boldsymbol{v}\rho)$ is a continuous function, by the so-called Fundamental Lemma of Calculus of Variations (see Brezis (2011), for instance) we infer that $\partial_t \rho + \nabla \cdot (\boldsymbol{v}\rho) = 0$ for any $(t, \boldsymbol{x}) \in (t_0, T) \times \Omega$ and $\rho_{t_0}(\boldsymbol{x}) = \rho(t_0, \boldsymbol{x})$ for any $\boldsymbol{x} \in \Omega$, thus recovering the pointwise formulation of Eq. (4.1).

Solving the Continuity Equation. The proof of the first part of Theorem 5 is a corollary of the following general statement.

Theorem 8. *Let $\Omega \subset \mathbb{R}^d$ be an open set and let $0 \le t_0 < T$. Let $\boldsymbol{v} : [t_0, T] \times \Omega \to \mathbb{R}^d$ be a globally bounded velocity field that satisfies Eq. (B.5). Assume that the flow of $\boldsymbol{v}$ starting at time t_0, denoted by $\boldsymbol{X}$, is well defined in $[t_0, T] \times \Omega$ and that $\boldsymbol{X}_t : \Omega \to \Omega$ is a bilipschitz transformation of Ω. Letting $\rho(\boldsymbol{x}, t)$ be defined by Eq. (B.2), then ρ is a distributional solution to the continuity equation Eq. (4.1) according to Definition 7.*

We remark that, under the assumptions of Theorem 5, the flow map $\boldsymbol{X}$ is given and we build velocity field $\boldsymbol{v}$ that has $\boldsymbol{X}$ as a unique flow map. Hence, by construction, Ω is invariant under $\boldsymbol{X}_t$ for any $t \in [t_0, T]$. Thus, the assumptions of Theorem 8 are satisfied.

Proof of Theorem 8. By Theorem 6, the flow map $\boldsymbol{X}$ starting at time t_0 associated to $\boldsymbol{v}$ is well defined. Hence, defining ρ by Eq. (B.2), for any $t \in [t_0, T]$ we have that $\rho(t, \cdot)$ is defined almost everywhere in $\mathbb{R}^d$. We check that ρ is a distributional solution to Eq. (4.1) according to Definition 7. To begin, we check that $\rho \in L^\infty([t_0, T]; L^1(\mathbb{R}^d))$. Indeed, after the change of variables $\boldsymbol{X}_t^{-1}(\boldsymbol{x}) = \boldsymbol{y}$, using the Area formula with $dy = |\det J\boldsymbol{X}_t^{-1}(\boldsymbol{x})| dx$, we have that

$$\int_\Omega |\rho(t, \boldsymbol{x})| \, \mathrm{d}x = \int_\Omega |\rho_{t_0}(\boldsymbol{X}_t^{-1}(\boldsymbol{x}))| |\det J\boldsymbol{X}_t^{-1}(\boldsymbol{x})| \, \mathrm{d}x = \int_\Omega |\rho_{t_0}(\boldsymbol{y})| \, \mathrm{d}y.$$

Therefore, the total mass is preserved along the time evolution. Then, fix a test function $\phi \in C_c^\infty([t_0, T) \times \Omega)$ and, performing again the change of variables $\boldsymbol{y} = \boldsymbol{X}_t^{-1}(\boldsymbol{x})$, we have that

$$\begin{aligned}
&\int_{t_0}^T \int_\Omega [\partial_t \phi + \boldsymbol{v} \cdot \nabla \phi] \rho \, \mathrm{d}x \, \mathrm{d}t \\
&\quad = \int_{t_0}^T \int_\Omega [\partial_t \phi(t, \boldsymbol{x}) + \boldsymbol{v}(t, \boldsymbol{x}) \cdot \nabla(t, \boldsymbol{x}) \phi] \rho_{t_0}(\boldsymbol{X}_t^{-1}(\boldsymbol{x})) |\det J\boldsymbol{X}_t^{-1}(\boldsymbol{x})| \, \mathrm{d}x \, \mathrm{d}t \\
&\quad = \int_{t_0}^T \int_\Omega [\partial_t \phi(t, \boldsymbol{X}_t(\boldsymbol{y})) + \boldsymbol{v}(t, \boldsymbol{X}_t(\boldsymbol{y})) \cdot \nabla \phi(t, \boldsymbol{X}_t(\boldsymbol{y}))] \rho_{t_0}(\boldsymbol{y}) \, \mathrm{d}y \, \mathrm{d}t.
\end{aligned}$$

Recalling that $\boldsymbol{X}_t$ is the flow map generated by the velocity field $\boldsymbol{v}$ starting at time t_0, the latter is equal to

$$\begin{aligned}
&\int_{t_0}^T \int_\Omega [\partial_t \phi(t, \boldsymbol{X}_t(\boldsymbol{y})) + \partial_t \boldsymbol{X}_t(\boldsymbol{y}) \cdot \nabla \phi(t, \boldsymbol{X}_t(\boldsymbol{y}))] \rho_{t_0}(\boldsymbol{y}) \, \mathrm{d}y \, \mathrm{d}t \\
&\qquad = \int_\Omega \rho_{t_0}(\boldsymbol{y}) \int_{t_0}^T \frac{\mathrm{d}}{\mathrm{d}t} \phi(t, \boldsymbol{X}_t(\boldsymbol{y})) \, \mathrm{d}t \, \mathrm{d}y,
\end{aligned}$$

after using Fubini's Theorem and the chain rule for the derivatives. Thus, by the Fundamental Theorem of Calculus, we have that

$$\begin{aligned}\int_\Omega \rho_{t_0}(\boldsymbol{y}) \int_{t_0}^{T} \frac{\mathrm{d}}{\mathrm{d}t}\phi(t, \boldsymbol{X}_t(\boldsymbol{y}))\ \mathrm{d}t\ \mathrm{d}y &= \int_\Omega \rho_{t_0}(\boldsymbol{y})[\phi(T, \boldsymbol{X}_T(\boldsymbol{y})) - \phi(0, \boldsymbol{X}_{t_0}(\boldsymbol{y}))]\ \mathrm{d}y \\ &= -\int_\Omega \phi(t_0, \boldsymbol{y})\rho_{t_0}(\boldsymbol{y})\ \mathrm{d}y,\end{aligned}$$

since $\phi(T, \cdot) \equiv 0$ and $\boldsymbol{X}_{t_0}$ is the identity map. □

Finally, we are able to conclude the proof of Theorem 5.

Conclusion of the proof of Theorem 5. We discussed the fact that ρ is a pointwise solution to the continuity equation Eq. (4.1) in Remark 2. Indeed, by the explicit formula Eq. (B.2) it is clear that ρ is $C^\infty([t_0, T) \times \Omega)$ and that ρ is nonnegative whenever ρ_{t_0} is nonnegative. Thus, we check that Eq. (B.4) is satisfied. Indeed, by the chain rule we have that

$$\begin{aligned}\frac{d}{dt}\ln(\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))) &= \frac{1}{\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))}\left[\partial_t\rho(t, \boldsymbol{X}_t(\boldsymbol{x})) + \nabla\rho(t, \boldsymbol{X}_t(\boldsymbol{x})) \cdot \frac{d}{dt}\boldsymbol{X}_t(\boldsymbol{x})\right] \\ &= \frac{1}{\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))}\left[\partial_t\rho(t, \boldsymbol{X}_t(\boldsymbol{x})) + \nabla\rho(t, \boldsymbol{X}_t(\boldsymbol{x})) \cdot \boldsymbol{v}(t, \boldsymbol{X}_t(\boldsymbol{x}))\right] \\ &= \frac{1}{\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))}\Big[\partial_t\rho(t, \boldsymbol{X}_t(\boldsymbol{x})) + \nabla \cdot (\boldsymbol{v}(t, \boldsymbol{X}_t(\boldsymbol{x}))\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))) \\ &\qquad - (\nabla \cdot \boldsymbol{v}(t, \boldsymbol{X}_t(\boldsymbol{x})))\rho(t, \boldsymbol{X}_t(\boldsymbol{x}))\Big] \\ &= -\nabla \cdot \boldsymbol{v}(t, \boldsymbol{X}_t(\boldsymbol{x})),\end{aligned}$$

since ρ satisfies the continuity equation at any point $(t, \boldsymbol{x}) \in (t_0, T) \times \Omega$. □

B.2 CALCULATING THE DENSITY AND VELOCITY

In this subsection we explicitly provide the steps to get to the analytical expressions for the velocity and density in Eq. (4.13) and Eq. (4.14).

B.2.1 *Calculating the Density*

We now report explicitly the steps to get to Eq. (4.13).

Let f and g be diffeomorphisms on $\mathbb{R}^d$, and A and B positive definite matrices. We denote with $J(f)(g(x))$ the Jacobian of f evaluated at the point $g(x)$. In the following part we make use of identities that follow from the chain rule, the inverse function theorem, and properties of the determinant:

$$J(f \circ g)(x) = J(f)(g(x))\; J(g)(x), \tag{B.12}$$

$$|\det J(f)(x)| = \left|\det J(f^{-1})(f(x))\right|^{-1}, \tag{B.13}$$

$$|\det(A \cdot B)| = |\det(A)| \cdot |\det(B)|\,. \tag{B.14}$$

$$\begin{aligned}\hat{\rho}(t,x) &= \hat{\rho}_{t_0}(\hat{X}_t^{-1}(x))|\det J(\hat{X}_t^{-1})(x)| \\ &= \hat{\rho}_{\text{base}}\Big((\Phi_{t_0} \circ \hat{X}_t^{-1})(x)\Big)\left|\det J(\Phi_{t_0})(\hat{X}_t^{-1}(x))\right|\left|\det J(\hat{X}_t^{-1})(x)\right|\end{aligned} \tag{B.15}$$

$$\begin{aligned}\hat{\rho}_{\text{base}}\Big((\Phi_{t_0} \circ \hat{X}_t^{-1})(x)\Big) &= \hat{\rho}_{\text{base}}\Big((\Phi_{t_0} \circ \Phi_{t_0}^{-1} \circ \Phi_t)(x)\Big) \\ &= \hat{\rho}_{\text{base}}(\Phi_t(x))\end{aligned} \tag{B.16}$$

$$\begin{aligned}\left|\det J(\Phi_{t_0})(\hat{X}_t^{-1}(x))\right| &= \left|\det J(\Phi_{t_0})\Big((\Phi_{t_0}^{-1} \circ \Phi_t)(x)\Big)\right| \\ &= \left|\det J(\Phi_{t_0}^{-1})\Big((\Phi_{t_0} \circ \Phi_{t_0}^{-1} \circ \Phi_t)(x)\Big)\right|^{-1} \\ &= \left|\det J(\Phi_{t_0}^{-1})(\Phi_t(x))\right|^{-1}\end{aligned} \tag{B.17}$$

$$\begin{aligned}\left|\det J(\hat{X}_t^{-1})(x)\right| &= \left|\det J(\Phi_{t_0}^{-1} \circ \Phi_t)(x)\right| \\ &= \left|\det J(\Phi_{t_0}^{-1})(\Phi_t(x))\right| \cdot |\det J(\Phi_t)(x)|\end{aligned} \tag{B.18}$$

Combing the three terms we get:

$$\begin{aligned} \hat{\rho}(t,x) &= \hat{\rho}_{t_0}(\hat{X}_t^{-1}(x))|\det J(\hat{X}_t^{-1})(x)| && \text{(B.19)} \\ &= \hat{\rho}_{\text{base}}(\Phi_t(x)) \cdot 1 \cdot |\det J(\Phi_t)(x)| && \text{(B.20)} \\ &= \hat{\rho}_{\text{base}}(\Phi_t(x))\, |\det J(\Phi_t)(x)| && \text{(B.21)} \end{aligned}$$

where we used in Eq. (B.20) the identity

$$\left|\det J(\Phi_{t_0}^{-1})(\Phi_t(x))\right|^{-1} \cdot \left|\det J(\Phi_{t_0}^{-1})(\Phi_t(x))\right| = 1 \tag{B.22}$$

B.2.2 *Calculating the Velocity without Inverting the Flow.*

We now report explicitly the steps to get to Eq. 4.14. Firstly, we show that the velocity can be expressed in terms of the flow bijection Φ_t and its inverse Φ_t^{-1}.

$$\begin{aligned} \hat{v}(t,x) &= \frac{\partial \hat{X}_t}{\partial t}\left(\hat{X}_t^{-1}(x)\right) && \text{(B.23)} \\ &= \frac{\partial(\Phi_t^{-1} \circ \Phi_{t_0})}{\partial t}\left((\Phi_{t_0}^{-1} \circ \Phi_t)(x)\right) && \text{(B.24)} \\ &= \frac{\partial(\Phi_t^{-1})}{\partial t}\left(\underbrace{(\Phi_{t_0} \circ \Phi_{t_0}^{-1}}_{=1} \circ \Phi_t)(x)\right) && \text{(B.25)} \\ &= \frac{\partial \Phi_t^{-1}}{\partial t}\left(\Phi_t(x)\right) && \text{(B.26)} \end{aligned}$$

In a second step, the velocity can be written without the need for explicit inversion of the bijective layer Φ_t, allowing for efficient computation in practice.

Let Φ_t and Φ_t^{-1} be the maps from x_t to z and vice versa respectively.

$$\Phi_t(x_t) = z, \quad \Phi_t^{-1}(z) = x_t \tag{B.27}$$

Clearly, $\Phi_t\left(\Phi_t^{-1}(z)\right) = z$ and z does not depend on time, i.e. $\frac{\mathrm{d}}{\mathrm{d}t}z = 0$. We can now explicitly compute the total derivative and find a formulation for the velocity that requires inverting a Jacobian instead of inverting the map Φ_t.

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\Phi_t\left(\Phi_t^{-1}(z)\right)\right) = \frac{\mathrm{d}}{\mathrm{d}t} z = 0 \tag{B.28}$$

$$\Rightarrow \frac{\partial \Phi_t}{\partial t}\big(\overbrace{\Phi_t^{-1}(z)}^{=x_t}\big) + \overbrace{\frac{\partial \Phi_t}{\partial x_t}\left(\Phi_t^{-1}(z)\right)}^{=J\Phi_t(x_t)} \frac{\partial \Phi_t^{-1}}{\partial t}\big(\overbrace{\ z\ }^{\Phi_t(x_t)}\big) = 0 \tag{B.29}$$

$$\Rightarrow \frac{\partial \Phi_t^{-1}}{\partial t}\left(\Phi_t(x_t)\right) = -\left[J\Phi_t(x_t)\right]^{-1} \frac{\partial \Phi_t}{\partial t}(x_t) \tag{B.30}$$

B.3 IMPLEMENTATION

This section provides high-level descriptions of the implementations of the models used in all experiments. For details on experiment-specific implementations, we refer to the corresponding sections B.4.1 to B.4.3.

LFlows. Our code for LFlows is based on the *nflows* library for bijective neural networks (Durkan et al., 2020). We extended *nflows* by providing a range of additional (conditional) transformations, such as invertible densenets with sinusoidal activation, in addition to the modifications required for the LFlows. The LFlow code is provided as supplementary material of our work (Torres et al., 2024).

SLDA. We represent the density ρ_{t_0} at the departure time t_0 with a bilinear interpolation of a learnable equidistant mesh in 2D or 3D. We parameterize the velocity with a neural network with smooth activation functions. In 2D settings, we evaluate the divergence in Eq. (4.4) exactly via autograd, whereas we stochasticly estimate it in 3D during training (Grathwohl et al., 2019). To solve multiple ODEs with different start times in parallel, we leverage the rescaling trick discussed in Appendix F of R. T. Chen et al. (2020). In all settings we use a Dormand-Prince solver of order 5 with absolute and relative tolerance of 1e-5. The adjoint is computed with the *torchdiffeq* library (R. T. Q. Chen, 2018).

DFNNs. We rely on the vector-based parameterization of DFNNs given in Section 7.1 of Richter-Powell et al. (2022), which ensures non-negative densities. We use the code provided by the authors.

PINNs. We implement standard PINNs without additional resampling schemes or loss terms, and use either ReLu (Section 4.5.1) or sinusoidal activations (Sitzmann et al., 2020) (Section 4.5.3). The collocation points are resampled each training iteration with quasi-random Sobol sequences.

B.4 ADDITIONAL INFORMATION ON THE EXPERIMENTS

B.4.1 *Experiment: Simulated Densities*

Data Generation. We generate the data by defining a Lagrangian flow map for an initial unnormalized Gaussian mixture density. The initial density of the simulated problem is based on a mixture of 4 independent Gaussians arranged around the origin with varying radii and a standard deviation of 0.1. We defined the density to be restricted to $\Omega = (-4, 4)^d$ with $(\rho \boldsymbol{v})|_{\delta\Omega} = 0$:

$$\rho_0(\boldsymbol{x}) = \begin{cases} \sum_{i=1}^{4} \frac{1}{4} \mathcal{N}(\boldsymbol{\mu_i}, 0.1\boldsymbol{I}) & \text{if } \boldsymbol{x} \in (-4, 4)^d \\ 0 & \text{else} \end{cases} \tag{B.31}$$

The changes in density on the xy axes are simulated by directly parameterizing the Lagrangian solution map $\boldsymbol{X}_t(\boldsymbol{x}_0) : (-4, 4)^2 \mapsto (-4, 4)^2$ for $t \in [0, 1.2]$:

$$\boldsymbol{X}_t(\boldsymbol{x}_0) = 4 \cdot \tanh\left(\frac{(0.5t + 1)}{4} A_{rot}(t) A_{scale}(t) \left(4 \cdot \operatorname{atanh}(0.25\boldsymbol{x}_0) + \operatorname{shift}(t)\right)\right) \tag{B.32}$$

with

$$A_{rot}(t) = \begin{bmatrix} \cos(2\pi t) & -\sin(2\pi t) \\ \sin(2\pi t) & \cos(2\pi t) \end{bmatrix} \tag{B.33}$$

$$A_{scale}(t) = \begin{bmatrix} 1 + 0.1t & 0 \\ 0 & 1 + 0.1t \end{bmatrix} \tag{B.34}$$

$$\operatorname{shift}(t) = \sin(\pi t) \begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix} \tag{B.35}$$

In 3D, the z component of $\boldsymbol{X}_t$ is always an identity map, limiting the dynamics to the xy axis.

The density and velocity are then given by:

$$\rho(t, \boldsymbol{x}) = \rho_0\left(\boldsymbol{X}_t^{-1}(\boldsymbol{x})\right) |\det J\boldsymbol{X}_t^{-1}(\boldsymbol{x})| \tag{B.36}$$

$$\boldsymbol{v}(t, \boldsymbol{x}) = \frac{\partial \boldsymbol{X}_t}{\partial t}(\boldsymbol{X}_t^{-1}(\boldsymbol{x})) \tag{B.37}$$

where all involved derivatives are computed using automatic differentiation. Observations are available at 21 equidistant timesteps in the range $[0, 1]$. The test data set covers the time range $[0, 1.2]$. To simulate noisy measurements, additional Gaussian noise is added to the observed velocities and log densities during training.

B.4.1.1 *Training and Architecture details*

Hyperparameter Optimization. We optimize each model based on the explained variance (R^2) of the density on validation data. Firstly, we manually performed a general architecture selection. Subsequently, we tuned parameters such as the number of layers, units, learning rate, loss- and regularization weights with the black-box optimization framework Optuna [1](Akiba et al., 2019), using the default Tree-structured Parzen Estimator as sampler. We trained the LFlows and PINNs on a minibatch size of 16384 and the SLDA on a minibatch size of 4096. As the 2nd order derivatives of the DFNNs require a lot of GPU memory, DFNNs were limited to a minibatch size of 2048. For the optimized hyperparameters of each model, we refer to the code provided in the supplementary material of Torres et al. (2024).

LFlows. For the initial layer we first rescale the domain and then use a *atanh* bijection for restricting the domain to $[-4, 4]^d$. For the remaining layers we used blocks consisting of invertible Dense Nets (i-DenseNet) (Perugachi-Diaz et al., 2021) followed by SVD layers conditioned on the embedding. For the i-DenseNet we rely on concatenated sinusoidal activations (CSin) with a ω of 15. Each i-DenseNet has a depth of 3 and before each block, we use an Activation Normalization layer. We enforce the Lipschitz constant of the i-DenseNet to be 0.97.

DFNNs. The Divergence-Free Neural Networks do not directly provide access to the velocity $\boldsymbol{v}$, but only to the flux $\boldsymbol{F} = (\rho \boldsymbol{v})$. Hence, calculating the velocity $\boldsymbol{v} = \boldsymbol{F}/\rho$ in low-density regions leads to numerical issues. To avoid

1 https://optuna.org/

this, we train DFNNs directly on the flux instead of the velocity. Furthermore, we require non-negative densities, so we use the parameterization with subharmonic functions discussed in Section 7.1 of Richter-Powell et al. (2022). As the predicted densities can still be zero, we train the DFNNs on the MSE of the densities (instead of log densities).

PINNs. To facilitate training of the PINN, we use sinusoidal activation functions as presented by (Sitzmann et al., 2020). We use a frequency multiplier of $\omega_0 = 12$ in the first layer. Collocation points are sampled within the full domain, uniformly distributed in $[0, 1.2] \times \Omega$. Instead of a purely random sampler, we rely on quasi-random low-discrepancy samples obtained via Sobol Sequences. In each minibatch, 2^{16} collocation points are sampled. The minimized PDE loss is $L(t, \boldsymbol{x}) = ||\partial_t \rho(t, \boldsymbol{x}) + \nabla \cdot (\rho(t, \boldsymbol{x}) \boldsymbol{v}(t, \boldsymbol{x}))||^2$, averaged over the collocation points.

SLDA. Relative and absolute tolerances of the solver during hyperparameter optimization were 10^{-3} and for the final run with the tuned hyperparameters 10^{-5}. Lower tolerances during hyperparameter search were not feasible, as they significantly increased the runtime for the problem at hand. For stable dynamics, the hypernetworks provided by the code of Grathwohl et al. (2019) were necessary. These hyper networks are conditioned on time and provide a network taking the space coordinates as input, i.e. $\boldsymbol{v}_{\Theta(t)}(\boldsymbol{x})$ with Θ being the hyper network. Using instead fully connected layers (that still fulfill the smoothness requirements) led in our experience to difficult dynamics for the adaptive ODE solvers.

Boundary Conditions. For PINNs, SLDA, and DFNNs the boundary condition $\rho(\boldsymbol{x})\boldsymbol{v}(\boldsymbol{x})|_{\delta\Omega} = \boldsymbol{0}$ is enforced via an additional penalty on points sampled at the boundary. For LFlows, the boundary was enforced via a bijection to $\Omega \setminus \delta\Omega$.

Numerical Evaluation of Physical Consistency. We compare the predicted density $\hat{\rho}(t, \boldsymbol{x})$ with the numerical solution of the initial value problem uniquely defined by $\hat{\rho}(t_0, \boldsymbol{x})$, $\hat{\boldsymbol{v}}(t, \boldsymbol{x})$ and Eq. (4.4). We obtain the numerical solution with an 8th order Dormand-Prince ODE solver with an absolute and relative tolerance of 1e-5. We set $t_0 = 0$. For the locations where we evaluate the sMAPE, we consider 10 equidistant timesteps in $[0.1, 1.2]$. For each timestep, we first randomly sample locations in $(4, 4)^d$, and we then randomly subsample 2500 locations with groundtruth densities larger than a threshold of 0.1 in 2D or 0.01 in 3D. With this procedure we avoid regions with near-zero density.

Total Mass Regularization. We penalize the total mass of the system for all methods except PINNs. For the LFlows and SLDA, we penalize the learnable normalization constant. For the DFNNs, no equivalent to the normalization constant is available. We instead introduce the penalty at points sampled on the domain $[0, 1.2] \times \Omega$. For PINNs additional regularization is not necessary. This is due to the side effect of the PDE loss being numerically small for small densities, leading to an automatic built-in penalty for large total mass.

Computational Resources. Each individual experiment for the synthetic data was run on individual NVIDIA TITAN X GPUs (12GB VRAM), using 20 CPU cores and 20GB RAM. To speed up hyperparameter tuning, up to 8 experiments were run in parallel using a SLURM-based compute cluster.

B.4.2 *Experiment: Dynamical Optimal Transport*

In Figure B.1 and Figure B.2 interpolated densities are shown for the 8Gaussians↔Circles dataset and the Pinwheel↔8Gaussians dataset.

B.4.2.1 *Evaluating the Integral*

For evaluating the integral required for the objective in Eq. (4.19), we reuse the code and method provided by Richter-Powell et al. (2022). The integral is estimated via importance sampling, with the sampling distribution $q(t, \boldsymbol{x}) = q(t)q(\boldsymbol{x})$. Samples in time are drawn uniformly with $q(t) \sim \mathcal{U}(t_0, t_1)$. Samples in space are drawn from a uniform mixture

$$q(\boldsymbol{x}) = \frac{1}{3}p_{t_0}(\boldsymbol{x}) + \frac{1}{3}p_{t_1}(\boldsymbol{x}) + \frac{1}{3}\mathcal{U}_\Omega(\boldsymbol{x}) \tag{B.38}$$

with $\mathcal{U}_\Omega$ being the uniform distribution on the domain.

B.4.2.2 *Discrete estimate of W2*

For estimating the W_2^2 distance with a discrete reference method we rely on 50000 samples. In our setting, more samples lead to convergence issues of the numerical solver.

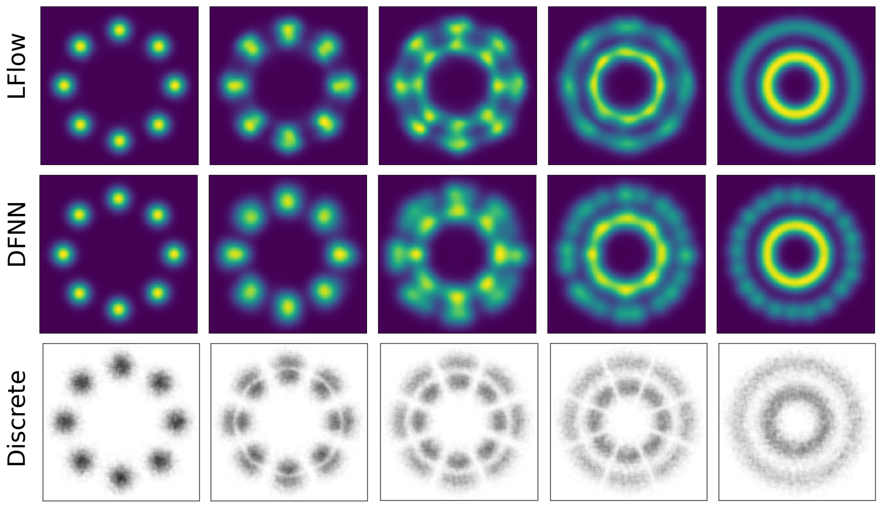


FIGURE B.1: Approximations of the 2D optimal transport map for the 8Gaussians↔Circles dataset with LFlows, DFNNs, and a discrete reference.

As the discrete W_2^2 estimate depends on the number of samples n, we explore an increasing number of samples and repeated runs. For each fixed n we repeat the estimate 10 times with different seeds and visualize the resulting box plots in B.3. We note that the differences with increasing sample size are comparatively small relative to the differences between the continuous methods in Section 4.5.2.

B.4.2.3 *Implementation*

LFlows. For LFlows we use 10 blocks of i-DenseNets, each preceded by an Activation Normalization layer. Each i-DenseNet has a depth of 5 with CSin activations using an ω of 15. We enforce the Lipschitz constant of the i-DenseNet to be 0.97. For the embedding of the time condition we use a residual neural network with 1 hidden layer, swish activations, and 128 hidden features. The dimension of the time embedding, i.e. the output of the embedding network, is of dimension 10. The model was trained with the ADAM optimizer for 5000 iterations with a learning rate of 2e-3 and 2048 data points per iteration.

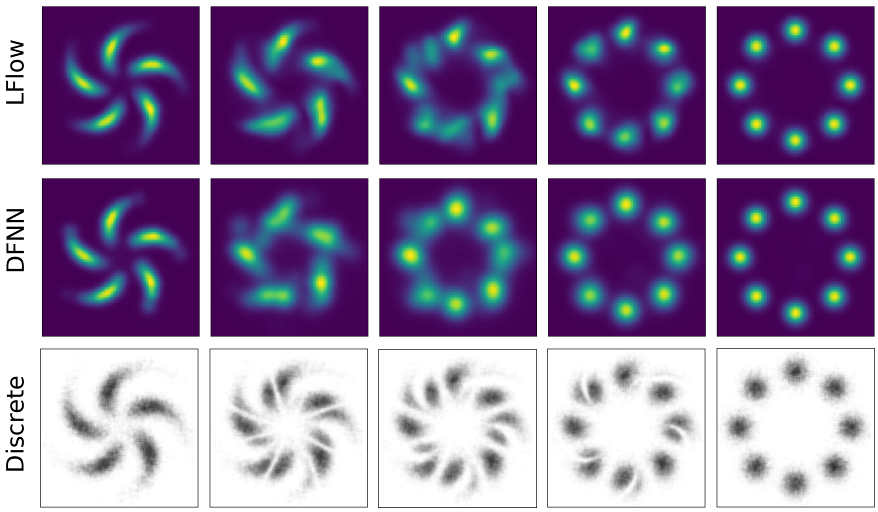


FIGURE B.2: Approximations of the 2D optimal transport map for the Pinwheel↔8Gaussians dataset with LFlows, DFNNs, and a discrete reference.

DFNNs. For DFNNs we directly used the experiment code provided by Richter-Powell et al. (2022). That is, the DFNNS are based on the parameterization with subharmonic functions discussed in Section 7.1 of Richter-Powell et al. (2022). The 128 mixtures of the subharmonic function are parameterized with a 4 layer neural network with 96 hidden features and swish activations. A fixed λ weight of 50 is set for the OT-penalty for all datasets. The DFNN is trained with the ADAM optimizer and a learning rate of 1e-3 over 10 000 iterations with 256 data points per iteration.

Computational Resources. The experiment was run on individual NVIDIA TITAN X GPUs (12GB VRAM), using 20 CPU cores and 20GB RAM.

B.4.3 *Experiment: Bird Migration*

About the data. The data provided by Nussbaumer et al. (2021) is originally based on weather radar measurements made available by the *European Operational Program for Exchange of Weather Radar Information* (EUMETNET/OPERA). The vertical profiles, i.e. density and velocity estimates at different altitude levels, were provided by the *European Network for the*

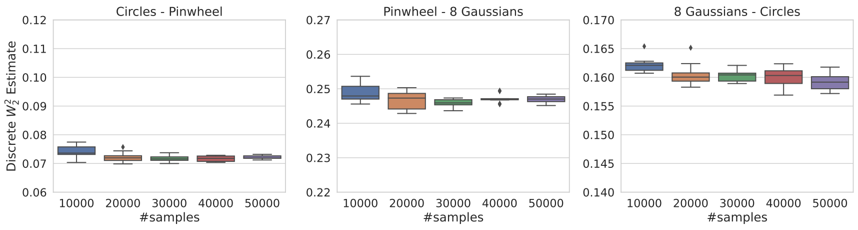


FIGURE B.3: Change of the discrete W_2^2 estimate based on increasing number of samples. Each boxplot is based on 10 repeated runs with different seeds.

Radar surveillance of Animal Movement (ENRAM), based on *vol2bird*[2], an algorithm for preprocessing raw radar scans. The raw data consists of volume scans from Doppler radars, measuring reflectivity and radial velocity of the surroundings. By filtering out biological and environmental scatters, it is possible to retain scans that mostly contain bird movements. Based on the reflectivity and radial velocity, the average bird density and velocity within a 15km radius is estimated for multiple altitude bins. For details about this process, we refer to Dokter et al. (2011). The final density and velocity measurements we use are openly available[3].

Preprocessing. The positions of the radars are given in the WGS84 coordinate reference system. We project it to the Cartesian reference system ETRS89-extended (EPSG:3035), effectively projecting longitude/latitude to x and y coordinates given in meters. As an additional preprocessing step, we excluded velocities that were measured together with a near-zero density. To generate the data set used in our experiment, we directly concatenated multiple nights and remove the daytime during which no measurements are available. In the supplementary material of Torres et al. (2024) we provide code for downloading and preprocessing the data.

Numerical Evaluation of Physical Consistency. We compare the predicted density $\hat{\rho}(t, \boldsymbol{x})$ with the numerical solution of the initial value problem uniquely defined by $\hat{\rho}(t_0, \boldsymbol{x})$, $\hat{\boldsymbol{v}}(t, \boldsymbol{x})$ and Eq. (4.4). The time of the earliest available datapoint within the three nights serves as t_0. We evaluated the error at a spatial equidistant grid of sidelength 50 in the xy dimension. The

2 https://github.com/adokter/vol2bird
3 https://zenodo.org/record/4587338/

z coordinate is randomly sampled for each of the grid entries. We evaluate the consistency loss at these xyz coordinates 10 time steps between the start and end of the three selected nights, and average the sMAPE over all $xyzt$ coordinates. For all models we use a 8th order Dormand-Prince ODE solver with an absolute and relative tolerance of 1e-5, which is one order higher than the ODE solver used for SLDA. The MLP has no z-component of the velocity, as no measurements of it are in the data. We thus set this component of the MLP to zero for computing the numerical ODE solution. The other models indirectly learn a z-component by enforcing the CE in 3D.

B.4.3.1 *Training and Architecture details*

LFlows. As first layer of the LFlow we use a *atanh* bijection that constrains Ω to a rectangular volume that is multiple times larger than the spatial extent of the radar positions. The following layers consist of 10 blocks of invertible Dense Nets (i-DenseNet) (Perugachi-Diaz et al., 2021), where we instead use CSin activations with an ω of 10. Each block has a depth of 5 and before each block we use an Activation Normalization layer. We enforce the Lipschitz constant of the i-DenseNet to be 0.97. We use a 2 layer residual network with a width of 128 for embedding the time before passing it to the i-DenseNet.

We initialized the log of the normalization constant $\ln(c)$ with 18.2 and set the weight of the total mass penalty to 1e-3. We trained for 50 epochs with a minibatch size of 16384 using the ADAM optimizer with a learning rate of 1e-2, a weight decay of 2e-3 and a cosine annealing learning rate schedule.

SLDA. For the SLDA we represent the initial density with a grid of size $20 \times 20 \times 20$. Values inbetween mesh points are evaluated with bilinear interpolation. We parameterize the velocity network with a hypernetwork as provided by the code of Grathwohl et al. (2019), similar to the simulated fluid flow experiment. The network consists of 5 layers with 512 hidden features and swish activations. We trained for 300 epochs with a minibatch size of 16384 using the ADAM optimizer with a learning rate of 1e-3, a weight decay of 5e-3 and a cosine annealing learning rate schedule.

DFNN. For the DFNN we use the parameterization for non-negative densities based on subharmonic functions (Section 7.1 Richter-Powell et al., 2022). The model consists of 4 layers with 256 hidden features and Swish activations, which parameterize 64 mixture components for the subharmonic function. We used a minibatch size of 2048 due to memory constraints. The model was trained for 100 epochs with a cosine annealing learning rate schedule.

MLP. We trained a 10 layer residual neural network with relu activations, intermediate batch normalization and 256 hidden features. The model was trained for 100 epochs with a minibatch size of 16384 using the ADAM optimizer with a learning rate of 1e-3, a weight decay of 1e-3 and a cosine annealing learning rate schedule.

PINN. For the PINN we use the same architecture as for the MLP, but with an additional PDE loss. We use 100 000 collocation points sampled from a quasi-random Sobol sequence and weight the PDE loss with 7e-4. The PINN was trained for 300 epochs with a minibatch size of 16384 using the ADAM optimizer with a learning rate of 1e-2, a weight decay of 1e-2, and a cosine annealing learning rate schedule

Computational Ressources. The experiment was run on an A100 GPU (40GB VRAM), using 20 CPUs and 30GB RAM. Repeated experiments were parallelized on a SLURM-based compute cluster.

B.4.3.2 *Predictions of all models*

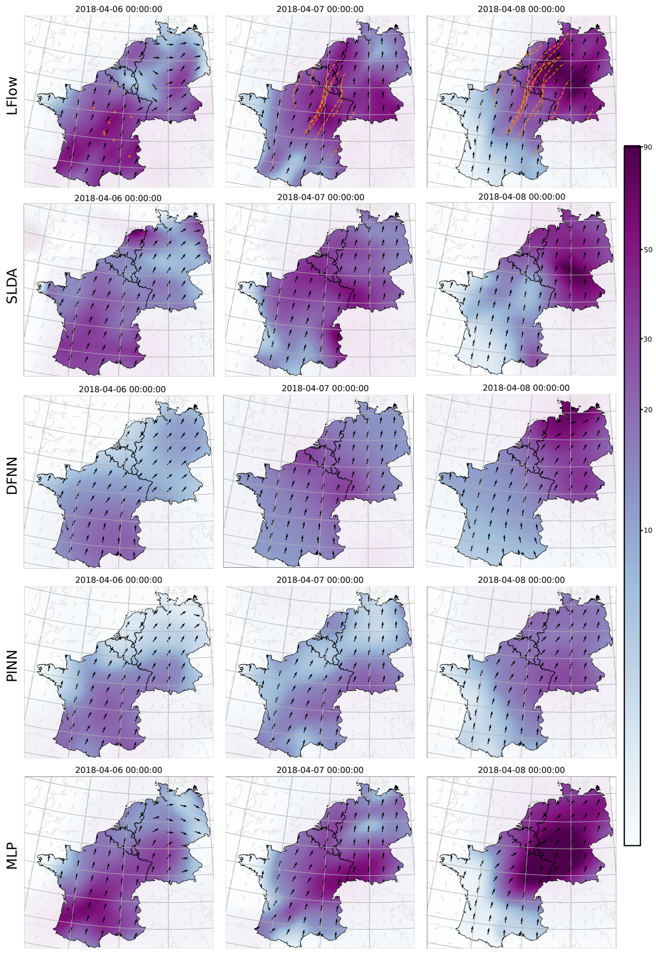


Figure B.4: Predicted density and normalized velocity integrated over z-axis for all considered models.

B.4.4 *Experiment: CSin for i-DenseNets*

B.4.4.1 *Qualitative Experiment for 2D Toy Data*

All flows were based on 10 i-DenseNet layers, each with a depth of 3 and a block-wise growth of 32. Inbetween the i-DenseNet layers we used activation normalization layers. The networks only differed with respect to the activation function.

We enforced a Lipschitz constant of 0.98 with spectral normalization. Before each i-DenseNet layer an activation normalization layer is added. The flow was trained for 10 000 iterations, with a minibatch size of 500. We used the ADAM optimizer with a default learning rate of 1e-3 and a weight decay of 1e-5. During training, the learning rate was decreased by a multiplicative factor of 0.9995 each iteration. Different learning rates led to qualitatively similar results when comparing the different activation functions.

B.4.4.2 *Quantitative evaluation on 2D Toy Data*

All flows were based on 10 i-DenseNet layers, each with a depth of 3 and a block-wise growth of 16, and just varied with respect to the activation function. No intermediate activation normalization was used to allow for a fair comparison with the CLipSwish results reported by Perugachi-Diaz et al. (2021). The remaining training parameters are identical to the qualitative 2D experiment desribed in Appendix B.4.4.1.

B.4.4.3 *Evaluation on UCI Data*

We rely on the preprocessed UCI data provided by (Papamakarios et al., 2017). A total of 30 i-DenseNet layers with intermediate activation normalization were used. Each i-DenseNet layer had a depth of 3, and a growth of 128. The CSin activations are set to $\omega = 15$. The model was trained with a batch size of 512 and trained for 400 000 iterations. The initial learning rate with the ADAM optimizer was set to 1e-3 with a multiplicative decay of 0.9999 per epoch, until a minimum learning rate of 5e-5 is reached.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

BIBLIOGRAPHY

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016a). {Tensorflow}: A system for {large-scale} machine learning. *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 265 (cit. on p. 21).

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016b). {Tensorflow}: A system for {large-scale} machine learning. *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 265 (cit. on p. 55).

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2623 (cit. on p. 120).

Ambrosio, L., & Crippa, G. (2008). Existence, uniqueness, stability and differentiability properties of the flow associated to weakly differentiable vector fields. In *Transport equations and multi-D hyperbolic conservation laws* (pp. 3–57, Vol. 5). Springer, Berlin. (Cit. on pp. 74, 113).

Ambrosio, L., & Crippa, G. (2014). Continuity equations and ode flows with non-smooth velocity. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, *144*(6), 1191 (cit. on p. 16).

Angell, R., & Sheldon, D. R. (2018). Inferring latent velocities from weather radar data using gaussian processes. *Advances in Neural Information Processing Systems*, *31* (cit. on p. 3).

Angriman, S., Cobelli, P., Mininni, P. D., Obligado, M., & Clark Di Leoni, P. (2023). Assimilation of statistical data into turbulent flows using physics-informed neural networks. *The European Physical Journal E*, *46*(3), 13 (cit. on p. 41).

Arbogast, T., & Bona, J. L. (1999). Methods of applied mathematics. *Department of Mathematics, University of Texas*, *2008* (cit. on p. 16).

Ardizzone, L., Lüth, C., Kruse, J., Rother, C., & Köthe, U. (2019). Guided image generation with conditional invertible neural networks. *arXiv preprint arXiv:1907.02392* (cit. on p. 35).

Arend Torres, F., Negri, M. M., Nagy-Huber, M., Samarin, M., & Roth, V. (2022). Mesh-free eulerian physics-informed neural networks. *arXiv preprint arXiv:2206.01545* (cit. on pp. 5, 12, 39, 45, 70, 103).

Asch, M., Bocquet, M., & Nodet, M. (2016). *Data assimilation: Methods, algorithms, and applications*. SIAM. (Cit. on p. 2).

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., & Jacobsen, J.-H. (2019). Invertible residual networks. *International Conference on Machine Learning*, 573 (cit. on pp. 28, 31, 32, 37).

Benamou, J.-D., & Brenier, Y. (2000). A computational fluid mechanics solution to the monge-kantorovich mass transfer problem. *Numerische Mathematik*, *84*(3), 375 (cit. on pp. 19, 83).

Berg, R. v. d., Hasenclever, L., Tomczak, J. M., & Welling, M. (2018). Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649* (cit. on p. 30).

Berkhahn, S., & Ehrhardt, M. (2022). A physics-informed neural network to model covid-19 infection and hospitalization scenarios. *Advances in Continuous and Discrete Models*, *2022*(1), 61 (cit. on p. 41).

Betancourt, M. (2017). A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434* (cit. on p. 54).

Biloš, M., Sommer, J., Rangapuram, S. S., Januschowski, T., & Günnemann, S. (2021). Neural flows: Efficient alternative to neural odes. *Advances in neural information processing systems*, *34*, 21325 (cit. on pp. 38, 72, 88).

Bonneel, N., Van De Panne, M., Paris, S., & Heidrich, W. (2011). Displacement interpolation using lagrangian mass transport. *Proceedings of the 2011 SIGGRAPH Asia conference*, 1 (cit. on p. 84).

Born, M., & Green, H. S. (1946). A general kinetic theory of liquids i. the molecular distribution functions. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, *188*(1012), 10 (cit. on p. 14).

Brezis, H. (2011). *Functional analysis, Sobolev spaces and partial differential equations*. Springer, New York. (Cit. on p. 113).

Cacuci, D. G. (1981a). Sensitivity theory for nonlinear systems. i. nonlinear functional analysis approach. *Journal of Mathematical Physics*, *22*(12), 2794 (cit. on pp. 24, 71).

Cacuci, D. G. (1981b). Sensitivity theory for nonlinear systems. ii. extensions to additional classes of responses. *Journal of Mathematical Physics*, *22*(12), 2803 (cit. on pp. 24, 71).

Carrillo, J. A., Craig, K., & Yao, Y. (2019). Aggregation-diffusion equations: Dynamics, asymptotics, and singular limits. *Active Particles, Volume 2: Advances in Theory, Models, and Applications*, 65 (cit. on p. 19).

Carrillo, J. A., Ranetbauer, H., & Wolfram, M.-T. (2016). Numerical simulation of nonlinear continuity equations by evolving diffeomorphisms. *Journal of Computational Physics*, *327*, 186 (cit. on p. 19).

Chen, J., Du, R., Li, P., & Lyu, L. (2019). Quasi-monte carlo sampling for machine-learning partial differential equations. *arXiv preprint arXiv:1911.01612* (cit. on p. 41).

Chen, R. T. Q. (2018). Torchdiffeq. (Cit. on p. 118).

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. *Advances in neural information processing systems*, *31* (cit. on pp. 24, 34, 71).

Chen, R. T., Amos, B., & Nickel, M. (2020). Neural spatio-temporal point processes. *International Conference on Learning Representations* (cit. on pp. 35, 118).

Chen, R. T., Behrmann, J., Duvenaud, D. K., & Jacobsen, J.-H. (2019). Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, *32* (cit. on pp. 32, 33).

Chilson, P. B., Stepanian, P. M., & Kelly, J. F. (2017). Radar aeroecology. *Aeroecology*, 277 (cit. on pp. 2, 68).

Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., Spergel, D., & Ho, S. (2020). Lagrangian neural networks. *arXiv preprint arXiv:2003.04630* (cit. on p. 71).

Dheeru, D., & Taniskidou, E. K. (2017). Uci machine learning repository (cit. on p. 91).

Diamantakis, M. (2013). The semi-lagrangian technique in atmospheric modelling: Current status and future challenges. *ECMWF Seminar in numerical methods for atmosphere and ocean modelling*, 183 (cit. on p. 71).

Diamantakis, M., & Magnusson, L. (2016). Sensitivity of the ecmwf model to semi-lagrangian departure point iterations. *Monthly Weather Review*, *144*(9), 3233 (cit. on p. 71).

Dissanayake, M., & Phan-Thien, N. (1994). Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, *10*(3), 195 (cit. on pp. 12, 42).

Dokter, A. M., Liechti, F., Stark, H., Delobbe, L., Tabary, P., & Holleman, I. (2011). Bird migration flight altitudes studied by a network of operational weather radars. *Journal of the Royal Society Interface*, *8*(54), 30 (cit. on pp. 2, 55, 125).

Duane, S., Kennedy, A. D., Pendleton, B. J., & Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*, *195*(2), 216 (cit. on p. 54).

Durkan, C., Bekasov, A., Murray, I., & Papamakarios, G. (2019). Neural spline flows. *Advances in neural information processing systems*, *32* (cit. on pp. 28, 76, 91, 92).

Durkan, C., Bekasov, A., Murray, I., & Papamakarios, G. (2020, Nov.). nflows: Normalizing flows in PyTorch. (Cit. on p. 118).

Earl, D. J., & Deem, M. W. (2005). Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, *7*(23), 3910 (cit. on p. 64).

Eastwood, E. (1967, Oct.). *Radar ornithology*. Methuen young books. (Cit. on p. 2).

Elfwing, S., Uchibe, E., & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, *107*, 3 (cit. on p. 76).

Finlay, C., Jacobsen, J.-H., Nurbekyan, L., & Oberman, A. (2020). How to train your neural ode: The world of jacobian and kinetic regularization. *International conference on machine learning*, 3154 (cit. on p. 35).

Flamary, R., Courty, N., Gramfort, A., Alaya, M. Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N., Gautheron, L., Gayraud, N. T., Janati, H., Rakotomamonjy, A., Redko, I., Rolet, A., Schutz, A., Seguy, V., Sutherland, D. J., . . . Vayer, T. (2021). Pot: Python optimal transport. *Journal of Machine Learning Research*, *22*(78), 1 (cit. on pp. 84, 103).

Freitag, M. A. (2020). Numerical linear algebra in data assimilation. *GAMM-Mitteilungen*, *43*(3), e202000014 (cit. on p. 39).

Fuentes, M., Van Doren, B. M., Fink, D., & Sheldon, D. (2023). Birdflow: Learning seasonal bird movements from ebird data. *Methods in Ecology and Evolution*, *14*(3), 923 (cit. on p. 69).

Gasteren, H. v., Holleman, I., Bouten, W., Loon, E. v., & Shamoun-baranes, J. (2008). Extracting bird migration information from c-band doppler weather radars. *Ibis*, *150*(4), 674 (cit. on p. 2).

Gebauer, N., Gastegger, M., & Schütt, K. (2019). Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. *Advances in neural information processing systems*, *32* (cit. on p. 10).

Geer, A. J. (2021). Learning earth system models from observations: Machine learning or data assimilation? *Philosophical Transactions of the Royal Society A*, *379*(2194), 20200089 (cit. on p. 2).

Gingold, R. A., & Monaghan, J. J. (1977). Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, *181*(3), 375 (cit. on p. 48).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press. (Cit. on pp. xi, 10, 22).

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., & Duvenaud, D. (2019). Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations* (cit. on pp. 35, 91, 118, 121, 126).

Greydanus, S., Dzamba, M., & Yosinski, J. (2019). Hamiltonian neural networks. *Advances in neural information processing systems*, *32* (cit. on pp. 11, 71).

Gunasekar, S., Lee, J. D., Soudry, D., & Srebro, N. (2018). Implicit bias of gradient descent on linear convolutional networks. *Advances in neural information processing systems*, *31* (cit. on p. 10).

Ha, D., Dai, A. M., & Le, Q. V. (2017). Hypernetworks. *International Conference on Learning Representations* (cit. on p. 35).

Haber, E., & Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse problems*, *34*(1), 014004 (cit. on p. 24).

Hairer, E., Hochbruck, M., Iserles, A., & Lubich, C. (2006). Geometric numerical integration. *Oberwolfach Reports*, *3*(1), 805 (cit. on p. 99).

Hartman, P. (2002). *Ordinary differential equations* (Vol. 38) [Corrected reprint of the second (1982) edition [Birkhäuser, Boston, MA; MR0658490 (83e:34002)], With a foreword by Peter Bates]. Society for Industrial; Applied Mathematics (SIAM), Philadelphia, PA. (Cit. on pp. 17, 74, 111).

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, 1026 (cit. on p. 22).

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770 (cit. on p. 22).

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity mappings in deep residual networks. *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, 630 (cit. on p. 22).

Hersbach, H., Bell, B., Berrisford, P., Hirahara, S., Horányi, A., Muñoz-Sabater, J., Nicolas, J., Peubey, C., Radu, R., Schepers, D., et al. (2020). The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, *146*(730), 1999 (cit. on p. 71).

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (cit. on p. 10).

Hirsch, C. (2007). *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*. Elsevier. (Cit. on pp. 12, 15, 100).

Hoffman, M. D., Gelman, A., et al. (2014). The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, *15*(1), 1593 (cit. on p. 64).

Hoogeboom, E., Van Den Berg, R., & Welling, M. (2019). Emerging convolutions for generative normalizing flows. *International conference on machine learning*, 2771 (cit. on p. 30).

Hoover, W. G., & Hoover, C. (2003). Links between microscopic and macroscopic fluid mechanics. *Molecular Physics*, *101*(11), 1559 (cit. on p. 49).

Horn, J. W., & Kunz, T. H. (2008). Analyzing nexrad doppler radar images to assess nightly dispersal patterns and population trends in brazilian free-tailed bats (tadarida brasiliensis). *Integrative and Comparative Biology*, *48*(1), 24 (cit. on p. 2).

Hutchinson, M. F. (1989). A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, *18*(3), 1059 (cit. on pp. 32, 35).

Jagtap, A. D., Kharazmi, E., & Karniadakis, G. E. (2020). Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, *365*, 113028 (cit. on pp. 41, 70).

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (cit. on p. 21).

Kalnay, E., Li, H., Miyoshi, T., Yang, S.-C., & Ballabrera-Poy, J. (2007). 4-d-var or ensemble kalman filter? *Tellus A: Dynamic Meteorology and Oceanography*, *59*(5), 758 (cit. on p. 39).

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, *3*(6), 422 (cit. on pp. 1, 2, 11, 67).

Kidger, P. (2021). *On neural differential equations* [Doctoral dissertation, University of Oxford]. (Cit. on p. 25).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (cit. on p. 106).

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc. (Cit. on pp. 29, 76).

Knott, M., & Smith, C. S. (1984). On the optimal mapping of distributions. *Journal of Optimization Theory and Applications*, *43*(1), 39 (cit. on p. 103).

Kobyzev, I., Prince, S. J., & Brubaker, M. A. (2020). Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, *43*(11), 3964 (cit. on pp. 26, 28, 35).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25* (cit. on p. 21).

Kukačka, J., Golkov, V., & Cremers, D. (2017). Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686* (cit. on p. 10).

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, *9*(5), 987 (cit. on p. 41).

Lagaris, I. E., Likas, A. C., & Papageorgiou, D. G. (2000). Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, *11*(5), 1041 (cit. on p. 41).

Landau, L. D., & Lifshitz, E. M. (2013). *Fluid mechanics: Landau and lifshitz: Course of theoretical physics, volume 6* (Vol. 6). Elsevier. (Cit. on p. 15).

Lao, J., Suter, C., Langmore, I., Chimisov, C., Saxena, A., Sountsov, P., Moore, D., Saurous, R. A., Hoffman, M. D., & Dillon, J. V. (2020). Tfp. mcmc: Modern markov chain monte carlo tools built for modern hardware. *arXiv preprint arXiv:2002.01184* (cit. on p. 55).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436 (cit. on p. 21).

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., & Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, *1*(0) (cit. on p. 1).

Li, J., Song, Z., & Yang, B. (2023). Nvfi: Neural velocity fields for 3d physics learning from dynamic videos. *Thirty-seventh Conference on Neural Information Processing Systems* (cit. on p. 11).

Li, L., Hurault, S., & Solomon, J. M. (2023). Self-consistent velocity matching of probability flows. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing* *systems* (pp. 57038–57057, Vol. 36). Curran Associates, Inc. (Cit. on pp. 70, 74).

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020). Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895* (cit. on p. 70).

Liao, H., & He, J. (2021). Jacobian determinant of normalizing flows. (Cit. on p. 98).

Lind, S. J., Rogers, B. D., & Stansby, P. K. (2020). Review of smoothed particle hydrodynamics: Towards converged lagrangian flow modelling. *Proceedings of the Royal Society A*, *476*(2241), 20190801 (cit. on p. 48).

Lippert, F., Kranstauber, B., Forré, P. D., & van Loon, E. E. (2022). Learning to predict spatiotemporal movement dynamics from weather radar networks. *Methods in Ecology and Evolution*, *13*(12), 2811 (cit. on p. 68).

Lippert, F., Kranstauber, B., van Loon, E. E., & Forré, P. (2022). Physics-informed inference of aerial animal movements from weather radar data. *NeurIPS 2022 AI for Science: Progress and Promises* (cit. on p. 68).

Lu, L., Jin, P., & Karniadakis, G. E. (2019). Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193* (cit. on p. 70).

Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). Deepxde: A deep learning library for solving differential equations. *SIAM Review*, *63*(1), 208 (cit. on pp. 41, 49, 55, 64, 103).

Lu, Y., Zhong, A., Li, Q., & Dong, B. (2018). Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *International Conference on Machine Learning*, 3276 (cit. on p. 24).

Makkuva, A., Taghvaei, A., Oh, S., & Lee, J. (2020). Optimal transport mapping via input convex neural networks. *International Conference on Machine Learning*, 6672 (cit. on p. 84).

Márquez-Neila, P., Salzmann, M., & Fua, P. (2017). Imposing hard constraints on deep networks: Promises and limitations. *arXiv preprint arXiv:1706.02025* (cit. on p. 12).

Mathiasen, A., Hvilshøj, F., Rødsgaard Jørgensen, J., Nasery, A., & Mottin, D. (2020). What if neural networks had svds? *Advances in Neural Information Processing Systems*, *33*, 18411 (cit. on p. 30).

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., . . . Scopatz, A. (2017). Sympy: Symbolic computing in python. *PeerJ Computer Science*, *3*, e103 (cit. on p. 105).

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. *European conference on computer vision*, 405 (cit. on p. 44).

Miyato, T., Kataoka, T., Koyama, M., & Yoshida, Y. (2018). Spectral normalization for generative adversarial networks. *International Conference on Learning Representations* (cit. on pp. 10, 31, 32).

Nabian, M. A., Gladstone, R. J., & Meidani, H. (2021). Efficient training of physics-informed neural networks via importance sampling. *Computer-Aided Civil and Infrastructure Engineering*, *36*(8), 962 (cit. on pp. 49, 55).

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2021). Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, *2021*(12), 124003 (cit. on p. 21).

Negri, M. M., Torres, F. A., & Roth, V. (2023). Conditional matrix flows for gaussian graphical models. *Thirty-seventh Conference on Neural Information Processing Systems* (cit. on p. 28).

Nesterov, V., Wieser, M., & Roth, V. (2020). 3dmolnet: A generative network for molecular structures. *arXiv preprint arXiv:2010.06477* (cit. on p. 10).

Ning, X., Guan, J., Li, X.-A., Wei, Y., & Chen, F. (2023). Physics-informed neural networks integrating compartmental model for analyzing covid-19 transmission dynamics. *Viruses*, *15*(8), 1749 (cit. on p. 41).

Nussbaumer, R., Bauer, S., Benoit, L., Mariethoz, G., Liechti, F., & Schmid, B. (2021). Quantifying year-round nocturnal bird migration with a fluid dynamics model. *Journal of the Royal Society Interface*, *18*(179), 20210194 (cit. on pp. 2, 55, 68, 85, 124).

Nussbaumer, R., Benoit, L., Mariethoz, G., Liechti, F., Bauer, S., & Schmid, B. (2019). A geostatistical approach to estimate high resolution nocturnal bird migration densities from a weather radar network. *Remote Sensing*, *11*(19), 2233 (cit. on pp. 2, 3, 55, 59, 68).

Onken, D., Fung, S. W., Li, X., & Ruthotto, L. (2021). Ot-flow: Fast and accurate continuous normalizing flows via optimal transport. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*, 9223 (cit. on p. 35).

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, *22*(1), 2617 (cit. on pp. 26, 28, 98).

Papamakarios, G., Pavlakou, T., & Murray, I. (2017). Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, *30* (cit. on pp. 91, 129).

Parisi, D. R., Mariani, M. C., & Laborde, M. A. (2003). Solving differential equations with unsupervised neural networks. *Chemical Engineering and Processing: Process Intensification*, *42*(8-9), 715 (cit. on p. 43).

Parno, M. D., & Marzouk, Y. M. (2018). Transport map accelerated markov chain monte carlo. *SIAM/ASA Journal on Uncertainty Quantification*, *6*(2), 645 (cit. on p. 54).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., . . . Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. (Cit. on pp. 21, 55).

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., et al. (2022). Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214* (cit. on p. 1).

Perugachi-Diaz, Y., Tomczak, J., & Bhulai, S. (2021). Invertible densenets with concatenated lipswish. *Advances in Neural Information Processing Systems*, *34*, 17246 (cit. on pp. 32–34, 37, 76, 78, 91, 120, 126, 129).

Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, *7*(15), 510 (cit. on p. 29).

Pontryagin, L. S. (1987). *Mathematical theory of optimal processes*. CRC press. (Cit. on pp. 24, 71).

Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., & Courville, A. (2019). On the spectral bias of neural networks. *International Conference on Machine Learning*, 5301 (cit. on p. 44).

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. *Proceedings of the 26th annual international conference on machine learning*, 873 (cit. on p. 21).

Rainey, R. (1955). Observation of desert locust swarms by radar. *Nature*, *175*(4445), 77 (cit. on p. 2).

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686 (cit. on pp. 3, 11, 39, 41, 43, 48, 69, 77).

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561* (cit. on pp. 12, 43).

Recktenwald, G. W. (2004). Finite-difference approximations to the heat equation. *Mechanical Engineering*, *10*(01) (cit. on p. 60).

Rezende, D., & Mohamed, S. (2015). Variational inference with normalizing flows. *International conference on machine learning*, 1530 (cit. on p. 55).

Richter-Powell, J., Lipman, Y., & Chen, R. T. (2022). Neural conservation laws: A divergence-free perspective. *Advances in Neural Information Processing Systems* (cit. on pp. 70, 77, 83–85, 100, 118, 121, 122, 124, 126).

Robert, A. (1982). A semi-lagrangian and semi-implicit numerical integration scheme for the primitive meteorological equations. *Journal of the Meteorological Society of Japan. Ser. II*, *60*(1), 319 (cit. on p. 71).

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 10684 (cit. on p. 1).

Rudd, K. (2013). *Solving partial differential equations using artificial neural networks* [Doctoral dissertation, Duke University]. (Cit. on p. 41).

Ruthotto, L., & Haber, E. (2020). Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, *62*, 352 (cit. on p. 24).

Särkkä, S., & Solin, A. (2019). *Applied stochastic differential equations* (Vol. 10). Cambridge University Press. (Cit. on pp. 19, 20, 62, 63).

Schütt, K. T., Sauceda, H. E., Kindermans, P.-J., Tkatchenko, A., & Müller, K.-R. (2018). Schnet–a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, *148*(24) (cit. on p. 1).

Ser-Giacomi, E., Rossi, V., López, C., & Hernandez-Garcia, E. (2015). Flow networks: A characterization of geophysical fluid transport. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *25*(3) (cit. on p. 99).

Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., & Dahl, G. E. (2018). Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600* (cit. on p. 88).

Sharma, P., Chung, W. T., Akoush, B., & Ihme, M. (2023). A review of physics-informed machine learning in fluid mechanics. *Energies*, *16*(5), 2343 (cit. on p. 12).

Sirignano, J., & Spiliopoulos, K. (2018). Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, *375*, 1339 (cit. on pp. 40, 41, 43, 47).

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., & Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, *33*, 7462 (cit. on pp. 44, 53, 56, 78, 106, 119, 121).

Smolensky, P., et al. (1986). Information processing in dynamical systems: Foundations of harmony theory (cit. on p. 1).

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. *International conference on machine learning*, 2256 (cit. on p. 1).

Staniforth, A., & Côté, J. (1991). Semi-lagrangian integration schemes for atmospheric models—a review. *Monthly weather review*, *119*(9), 2206 (cit. on p. 71).

Stepanian, P. M., Wainwright, C. E., Frick, W. F., & Kelly, J. F. (2019). Weather surveillance radar as an objective tool for monitoring bat phenology and biogeography. *The Journal of Engineering*, *2019*(20), 7062 (cit. on p. 2).

Strang, G. (2022). *Introduction to linear algebra*. SIAM. (Cit. on p. 30).

Sturm, P. O., & Wexler, A. S. (2022). Conservation laws in a neural network architecture: Enforcing the atom balance of a julia-based photochemical model (vo. 2.0). *Geoscientific Model Development*, *15*(8), 3417 (cit. on p. 71).

Sukumar, N., & Srivastava, A. (2021). Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 114333 (cit. on p. 40).

Sullivan, B. L., Aycrigg, J. L., Barry, J. H., Bonney, R. E., Bruns, N., Cooper, C. B., Damoulas, T., Dhondt, A. A., Dietterich, T., Farnsworth, A., et al. (2014). The ebird enterprise: An integrated approach to development and application of citizen science. *Biological conservation*, *169*, 31 (cit. on p. 69).

Tadiparthi, V., & Bhattacharya, R. (2021). Optimal transport based refinement of physics-informed neural networks. *arXiv preprint arXiv: 2105.12307* (cit. on pp. 49, 55, 64, 103).

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., & Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, *33*, 7537 (cit. on pp. 44, 53).

Tomczak, J. M., & Welling, M. (2016). Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630* (cit. on p. 29).

Toner, J., & Tu, Y. (1995). Long-range order in a two-dimensional dynamical XY model: How birds fly together. *Physical review letters*, *75*(23), 4326 (cit. on pp. 49, 53).

Toner, J., & Tu, Y. (1998). Flocks, herds, and schools: A quantitative theory of flocking. *Physical review E*, *58*(4), 4828 (cit. on p. 53).

Tong, A., Huang, J., Wolf, G., Van Dijk, D., & Krishnaswamy, S. (2020). Trajectorynet: A dynamic optimal transport network for modeling cellular dynamics. *International conference on machine learning*, 9526 (cit. on pp. 11, 98).

Torres, F. A., Negri, M. M., Inversi, M., Aellen, J., & Roth, V. (2024). Lagrangian flow networks for conservation laws. *The Twelfth International Conference on Learning Representations* (cit. on pp. 5, 12, 25, 67, 109, 118, 120, 125).

Tu, Y., Toner, J., & Ulm, M. (1998). Sound waves and the absence of Galilean invariance in flocks. *Physical review letters*, *80*(21), 4819 (cit. on p. 53).

Uhlig, F. (2001). Constructive ways for generating (generalized) real orthogonal matrices as products of (generalized) symmetries. *Linear Algebra and its Applications*, *332*, 459 (cit. on p. 29).

Van Doren, B. M., & Horton, K. G. (2018). A continental system for forecasting bird migration. *Science*, *361*(6407), 1115 (cit. on pp. 2, 3).

van Milligen, B. P., Tribaldos, V., & Jiménez, J. (1995). Neural network differential equation and plasma equilibrium solver. *Physical review letters*, *75*(20), 3594 (cit. on p. 43).

Vapnik, V. (1999). *The nature of statistical learning theory*. Springer science & business media. (Cit. on p. 9).

Wang, S., Teng, Y., & Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, *43*(5), A3055 (cit. on p. 53).

Wang, S., Wang, H., & Perdikaris, P. (2021). On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, *384*, 113938 (cit. on p. 53).

Wang, S., Yu, X., & Perdikaris, P. (2022). When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, *449*, 110768 (cit. on p. 44).

Weinan, E. (2017). A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, *1*(5), 1 (cit. on p. 24).

Wessels, H., Weißenfels, C., & Wriggers, P. (2020). The neural particle method–an updated lagrangian physics informed neural network for computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, *368*, 113127 (cit. on p. 48).

Williams, T. C., Ireland, L. C., & Williams, J. M. (1973). High altitude flights of the free-tailed bat, tadarida brasiliensis, observed with radar. *Journal of Mammalogy*, *54*(4), 807 (cit. on p. 2).

Xu, Z.-Q. J., Zhang, Y., Luo, T., Xiao, Y., & Ma, Z. (2019). Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523* (cit. on p. 44).

Zhang, J., Lei, Q., & Dhillon, I. (2018). Stabilizing gradients for deep neural networks via efficient svd parameterization. *International Conference on Machine Learning*, 5806 (cit. on pp. 1, 30).