Physics-Informed Neural Networks for Power System Dynamics

Stiasny, Jochen Bernhard

Link to article, DOI:
10.11581/DTU.00000277

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

Division for Power and Energy Systems
Department of Wind and Energy Systems

Physics-Informed Neural Networks for Power System Dynamics

Jochen Stiasny

PhD Thesis, June 2023, Kongens Lyngby, Denmark

DANMARKS TEKNISKE UNIVERSITET
Division for Power and Energy Systems (PES)
DTU Wind and Energy Systems

Physics-Informed Neural Networks for Power System Dynamics

Dissertation, by Jochen Stiasny

Supervisors:
Associate Professor Spyros Chatzivasileiadis, Technical University of Denmark
Professor Pierre Pinson, Imperial College London

DTU - Technical University of Denmark, Kongens Lyngby - June 2023

Physics-Informed Neural Networks for Power System Dynamics

This thesis was prepared by:
Jochen Stiasny

Supervisors:
Associate Professor Spyros Chatzivasileiadis, Technical University of Denmark
Professor Pierre Pinson, Imperial College London

Dissertation Examination Committee:
Senior Researcher Tuhfe Göçmen
Department of Wind and Energy Systems, Technical University of Denmark, Denmark

Professor Michael (Misha) Chertkov
Department of Mathematics, University of Arizona, United States of America

Professor Damien Ernst
Department of Electrical Engineering and Computer Science, University of Liège, Belgium

Division for Power and Energy Systems (PES)
DTU Wind and Energy Systems
Elektrovej, Building 325
DK-2800 Kgs. Lyngby
Denmark

Release date: June 2023

Edition: 1.0

Class: Internal

Field: Electrical Engineering

Remarks: The dissertation is presented to the Department of Wind and Energy Systems of the Technical University of Denmark in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

ISBN: 000-00-00000-00-0

DOI: https://doi.org/10.11581/DTU.00000277

Your reason and your passion are the rudder and the sails of your seafaring soul. If either your sails or your rudder be broken, you can but toss and drift, or else be held at a standstill in mid-seas.

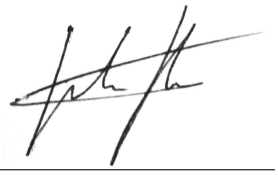
— Kahlil Gibran,
The Prophet

Preface

This thesis is prepared at the Department of Wind and Energy Systems of the Technical University of Denmark in partial fulfilment of the requirements for acquiring the degree of Doctor of Philosophy in Engineering. The Ph.D. project was supported by the multiDC project funded by Innovation Fund Denmark, Grant No. 6154-00020B, and an internal DTU scholarship.

This dissertation summarises the work carried out by the author during his Ph.D. project. It started on 15$^{\text{th}}$ January 2020, and it was completed on 30$^{\text{th}}$ June 2023. During this period, the author was hired by the Technical University of Denmark as a Ph.D. student at the Division for Power and Energy Systems (PES).

This thesis is composed of 9 chapters which summarise the work carried out over the course of the Ph.D. project.



Jochen Stiasny
June 2023

Acknowledgements

A seafaring soul – what a metaphor for pursuing a PhD. And to my great fortune, many people supported me in keeping my rudder and sails intact. I cannot thank you enough for it, let this be a start.

To begin, my gratitude goes to my supervisors, Spyros and Pierre. From the day I started my journey at DTU, I was sure that this was the place to be. The atmosphere in our research group has been so welcoming, inspiring, and respectful that I thought I had won the lottery. The role, the two of you played in this, cannot be understated – thank you for this remarkable experience! To be able to sail to new shores, there needs to be trust. Thank you, Spyros, for always placing this trust in me and my work! Your passion to think outside the box and to seek new opportunities is simply contagious, and ideal for exploring the world. Pierre, thank you for your guidance and putting things in perspective. It meant a lot to me for navigating a PhD and life beside it – you are a human lighthouse! My thanks also go to Baosen for hosting me in Seattle. It implied leaving safer waters for three months; our discussions and the inspiring change of scenery entirely justified it.

Sailing along with others is of course much more fun than alone. To all the people in ELMA, ELSY, PWR, EMA and my flatmates over these three and a half years: Spending time with you, at the office as well as outside of it, was always a source of joy, motivation, and energy. I hope, you have as many fond memories of this time as I do! What better place to share them than at a Friday bar or during an afternoon of boardgames – Eléa, Liyang, Linde, Yannick, David, Jan, I count on you! Sam, thank you for all the discussions and collaborations, brainstorming and working with you is just great fun! Peter, thank you for translating my abstract and to Daniel, Sam, and Anubhav, thank you for the invaluable feedback on this thesis. I would also like to thank two inspiring math teachers, Herrn Reiffert and Grant Sanderson (3blue1brown). On the topic of going back to high school times: Bartosz, Soso, Konrad, and Daniel, thank you for our friendship over all these years! Georgios, I still remember when you were showing me the office on the day of my interview ... what an amazing time followed, thank you for that! Ana, Daniel, and Tiago, I think it is time for another round of discussing until sunrise, it simply frees the mind! Andreas, Valeska, and Anubhav, thank you for all the trips and dinners throughout the years – I think (Indian) heaven on earth describes it best! Monsieur Peyo, thank you for all our short and long walks! Amandine, I cannot describe how glad I am that our paths crossed on this journey, experiencing and exploring this world with you is simply beyond words – just endless sparks of joy!

I believe, it is a blessing for every seafarer to know of a home port where one can rest, recharge, forge new plans, or simply enjoy a cup of coffee and a chat – Mutti, Vati, und Angelika, I could wish for no better home port!

Ahoi,
Jochen

Kgs. Lyngby, Denmark, June 2023

Table of Contents

List of Figures

List of Tables

Notation

General notation

$\boldsymbol{a}$	Vector / vector-valued function
$\bar{\boldsymbol{a}}$	Complex vector
$\hat{\boldsymbol{a}}$	Predicted / estimated value of $\boldsymbol{a}$
$\boldsymbol{a}^{(k)}$	k-th iteration or element in a set
a_i	i-th component of $\boldsymbol{a}$
$\boldsymbol{a}_i$	subset of $\boldsymbol{a}$ associated with the i-th subset (usually for generators)
A	Matrix
$\left\|\cdot\right\|_p$	L^p-norm
$\left\|\cdot\right\|_{T,p}$	L^p-norm in the dimensionless state space
$\Re$	Real part of a complex number
$\Im$	Imaginary part of a complex number
$\mathcal{O}$	Landau's symbol, indicates the order of an expression

Dynamical systems

t	Time
h	Time step size
$\frac{d}{dt}$	Temporal derivative
$\int \ldots dt$	Temporal integral
$\boldsymbol{x}$	System state
$\boldsymbol{x}_0$	Initial state ($\boldsymbol{x}(t_0)$)
$\tilde{\boldsymbol{x}}$	Dimensionless system state ($\tilde{\boldsymbol{x}} = \boldsymbol{T}(\boldsymbol{x})$)
$\boldsymbol{T}(\cdot)$	Transformation from the state space to the dimensionless state space
$\boldsymbol{u}$	Control input
$\boldsymbol{\lambda}$	System parameters
$\boldsymbol{f}$	System update function

Φ	Flow
ϕ_t	Flow map
$\boldsymbol{k}^j$	Runge-Kutta stage

Machine learning

$\mathcal{T}$	Task
$\mathcal{P}$	Performance metric
$\mathcal{E}$	Experience
h_θ	Hypothesis class parametrised by $\boldsymbol{\theta}$
$\mathcal{D}$	Dataset
$\mathcal{L}$	Loss
ℓ	Loss per data point
α	Loss term weight
$\boldsymbol{\theta}$	Neural network parameters
N_L	Number of layers
N_N	Number of neurons per layer
E	Epoch

Power system Differential Algebraic Equations

$\bar{\boldsymbol{v}}$	Complex voltage ($Ve^{j\theta}$)
$\bar{\boldsymbol{i}}^C$	Complex component current injections
$\bar{\boldsymbol{i}}^N$	Complex network current
$\bar{\boldsymbol{Y}}$	Complex network admittance matrix
$\boldsymbol{\Xi}$	Parameters of the approximated voltage
$\bar{\boldsymbol{\varepsilon}}$	Error in the current balance
$\boldsymbol{t}$	Query points
Δt	Time step size
$\boldsymbol{\rho}$	Residual, collection of $\bar{\boldsymbol{\varepsilon}}$ at the query points
J	Jacobian of $\boldsymbol{\rho}$ w.r.t. $\boldsymbol{\Xi}$

Acronyms

AD Automatic Differentiation.

BDF Backward Differentiation Formula.

BNN Bayesian Neural Network.

BPINN Bayesian Physics-Informed Neural Network.

BVTO Bias Variance Trade-off.

CPU Central Processing Unit.

DAE Differential Algebraic Equation.

ERK explicit Runge-Kutta.

FE Forward-Euler.

GPU Graphics Processing Unit.

HMC Hamiltonian Monte-Carlo.

HPC High-Perfomance Computing.

IRK implicit Runge-Kutta.

L-BFGS Limited Memory Broyden–Fletcher–Goldfarb–Shanno.

MAPE Maximum a Posteriori Estimator.

ML Machine Learning.

MLE Maximum Likelihood Estimator.

NeuralODE Neural Ordinary Differential Equation.

NN Neural Network.

ODE Ordinary Differential Equation.

OPF Optimal Power Flow.

PDE Partial Differential Equation.

PE Partitioned-Explicit.

PINN Physics-Informed Neural Network.

RK Runge-Kutta.

RK4 4th-order Runge-Kutta.

ROM Reduced-Order Modelling.

SAS Semi-Analytical Solution.

SciML Scientific Machine Learning.

SI Simulataneous-Implicit.

SINDy Sparse Identification of Non-Linear Dynamics.

SMIB Single-Machine Infinite-Bus.

TDS Time-Domain Simulation.

Abstract

The idea behind mathematical modelling is the representation of observations in a form that becomes useful for analysing, controlling, and predicting the underlying system. In the power system, modelling enables the operation of the power grid that aims for steady, secure, affordable, and carbon-neutral electricity supply. Achieving these objectives relies on having useful and tractable models. While the ongoing energy transition is a requirement to decarbonise the energy system, it, at the same time, poses a significant challenge for the existing modelling approaches and, hence, the operation of the grid. The need to consider more generation units, increased uncertainty, and more intricate dynamic behaviour calls for the development of complementary modelling approaches to better meet the operational objectives.

A key tool in power system operation is the solution of differential equations that govern the dynamic behaviour of the system. As their solution incurs a large computational cost and will increase in light of the energy transition, their acceleration is crucial for future operation. In this work, we investigate a recently suggested approach to solving these differential equations based on Machine Learning (ML), a method known as Physics-Informed Neural Networks (PINNs). We *learn* the solution and once it is learned, the evaluation can be several times faster than with a conventional numerical integration scheme. The question is if and how this approach can be applied to a realistic system size, i.e., are PINNs scalable?

We begin by providing a thorough investigation of the methodological proposition that PINNs offer in the context of a small dynamical system, i.e., a single machine. The introduction of a concept from the numerical analysis of dynamical systems, called *flow*, contributes to a better interpretation of the learned solution. On this basis, we demonstrate several techniques for improving the learned solution and to induce desirable numerical properties. To this end, we propose an additional loss term (dt-loss) and an extension to the architecture of PINNs (IRK-PINNs). The adoption of a probabilistic viewpoint of PINNs in the context of estimating the system parameters further enhances the understanding of PINNs as a modelling approach for dynamical systems.

Looking beyond a single machine system, the scalability to realistic sizes of the power grid with multiple machines proves to be challenging. While we show that it is possible to learn selected dynamics of a 39-bus system, the increasing complexity of the learning process becomes a practical barrier for the scalability of PINNs. Based on this analysis, we propose a simulator called *PINNSim* that combines the benefits of PINNs for small systems with the flexibility and scalability of a conventional numerical method. In PINNSim, the dynamics of each component are represented by a component-specific PINN, and we determine their interaction with a conventional root-finding algorithm. PINNSim allows for larger and hence fewer time steps as we demonstrate on a 9-bus system. It is conceptually scalable and enables a modular integration of PINNs.

Due to the safety critical nature of the power system, the adoption of ML-based modelling, such as PINNs, requires building trust on the grid operator's side. We reflect on this challenge and suggest a classification of ML-based methods based on the benefits they could provide for different tasks.

Resumé

Ideen bag matematisk modellering er repræsentationen af observationer i en form, der bliver nyttig til at analysere, kontrollere og forudsige det underliggende system. I elsystemet, muliggør modellering driften af elnettet, der sigter mod en stabil, sikker, overkommelig, og CO2-neutral elforsyning. At nå disse mål afhænger af at have nyttige og håndterbare modeller. Mens den igangværende energiomstilling er et krav for at dekarbonisere energien system, udgør det samtidig en væsentlig udfordring for de eksisterende modelleringstilgange og dermed driften af nettet. Behovet for at overveje flere produktionsenheder, stigende usikkerhed og mere indviklet dynamisk adfærd, kræver udvikling af komplementære modelleringsmetoder for bedre at opfylde disse operationelle mål.

Et nøgleværktøj i driften af elnettet er løsningen af differentialligninger, der beskriver systemets dynamiske adfærd. Da deres løsning medfører store beregningsomkostninger, som kun vil blive værre pga. energiomstillingen, er deres acceleration afgørende for fremtidig drift. I denne afhandling undersøger vi en nyligt foreslået tilgang til løsning af differentialligninger baseret på Machine Learning (ML), en metode kendt som Physics-Informed Neural Networks (PINNs). Vi lærer løsningen, og når den først er lært, kan evalueringen være flere gange hurtigere end den konventionelle numeriske integrationsordning. Spørgsmålet er, om og hvordan denne tilgang kan anvendes på en realistisk systemstørrelse, dvs. er PINN'er skalerbare?

Vi starter med at give en grundig undersøgelse af den metodiske baggrund, som PINNs tilbyder i sammenhæng med et lille dynamisk system, dvs. en enkelt maskine. Introduktion af et koncept fra den numeriske analyse af dynamiske systemer kaldet *flow* bidrager til en bedre fortolkning af den lærte løsning. På dette grundlag demonstrerer vi flere teknikker til at forbedre den lærte løsning og at inducere ønskelige numeriske egenskaber. Til dette formål foreslår vi et yderligere led i objektivfunktionen (dt-tab) og en udvidelse af PINNs (IRK-PINNs) arkitektur. Ved at tage et mere probabilistisk synspunkt for PINNs i forbindelse med yderligere estimering af systemparametrene øger forståelsen af PINNs som en modelleringstilgang til dynamiske systemer.

Ser man ud over et enkelt maskinsystem, så synes skalerbarheden til realistiske størrelser af elnettet med flere maskiner at være udfordrende. Mens vi viser, at det er muligt at lære nogle dynamikker i et 39-bus system, bliver den stigende kompleksitet af læreprocessen en praktisk barriere for skalerbarheden af PINNs. Baseret på denne analyse foreslår vi en simulator kaldet PINNSim der kombinerer fordelene ved PINNs til små systemer med fleksibiliteten og skalerbarheden af en konventionel numerisk metode. I PINNSim er dynamikken i hver komponent repræsenteret af en komponent specifik PINN, og vi bestemmer deres interaktion med en konventionel rod-søgende algoritme. PINNSim giver mulighed for større og dermed færre tidstrin, som vi demonstrerer på et 9-bus system. Det er konceptuelt skalerbart og muliggør en modulær integration af PINNs.

På grund af elsystemets sikkerhedskritiske karakter er brugen af ML-baseret modellering, som f.eks PINNs, noget der kræver opbygning af tillid på netoperatørens side. Vi beskriver denne

udfordring og foreslår en klassificering af ML-baserede metoder baseret på de fordele, de kunne give til forskellige opgaver.

CHAPTER 1

Introduction

1.1 Context and motivation

When we use models, we aim to build a mathematical representation of systems and processes that we experience through observations. Such systems can be natural, technical, societal, etc. – examples are the movement of planets in space, the power grid, or a population's reaction to a pandemic. By building a mathematical representation, models can help us to understand characteristics of these systems and make them amenable to analysis, prediction, control, and optimisation. In this thesis, the system in question is the power system, more specifically the power grid that connects loads and generation units and thereby satisfies the energy needs of a society. The task for grid operators and related actors is to ensure a safe and steady supply while being accessible and affordable for the entire population. In light of the need to decarbonise our energy use [1], the support of the ongoing energy transition places another important objective on power grid operation. The rapid development of renewable energy sources is a necessary condition to meet the objective of decarbonisation [2], but it has also major impacts on the operation and planning of the power grid.

To meet all constraints and objectives as well as possible, power grid operators need to continuously analyse the grid, react to disturbances and anticipate potentially critical operational scenarios in the future. To this end, power system researchers and the industry have developed many models, see, e.g., in [3]–[5]. Depending on the task at hand, these models will have different levels of *usefulness*. For example, for judging whether the system will remain within the operational limits, i.e., analysing the system stability, different models can inform about different stability phenomena [6], [7]; they might be considered accurate, but their usefulness depends on the task. At the same time, the evaluation of each model is associated with a computational cost. In Figure 1.1 we show these two dimensions in a schematic illustration. The solid black line represents the "model frontier", i.e., those models that provide the best trade-off between the conflicting objectives of usefulness and computational cost. Models that lie above this line, i.e., in the shaded areas can be used, but the models at the frontier would be cheaper.

In practice, this large region of available models is restricted from the left and the top: We only have limited computational resources and require a minimum level of usefulness from a model to solve the given task. The dotted lines illustrate these limits, and they demarcate a region of *applicable* models, that are sufficiently useful and do not exceed the computational budget. The model within this region that is the cheapest to evaluate yet solves the task would be the optimal model choice and is denoted by the star symbol. However, the constraints of the task, i.e., the computational budget and the required usefulness, can lead to a situation with no applicable model left – the unfilled circle in the upper right illustrates this situation. Out of necessity, one can adjust the budget or compromise on the requirement for being considered sufficiently useful. The two red circles show potential options. Increasing computational budget might be possible, however,

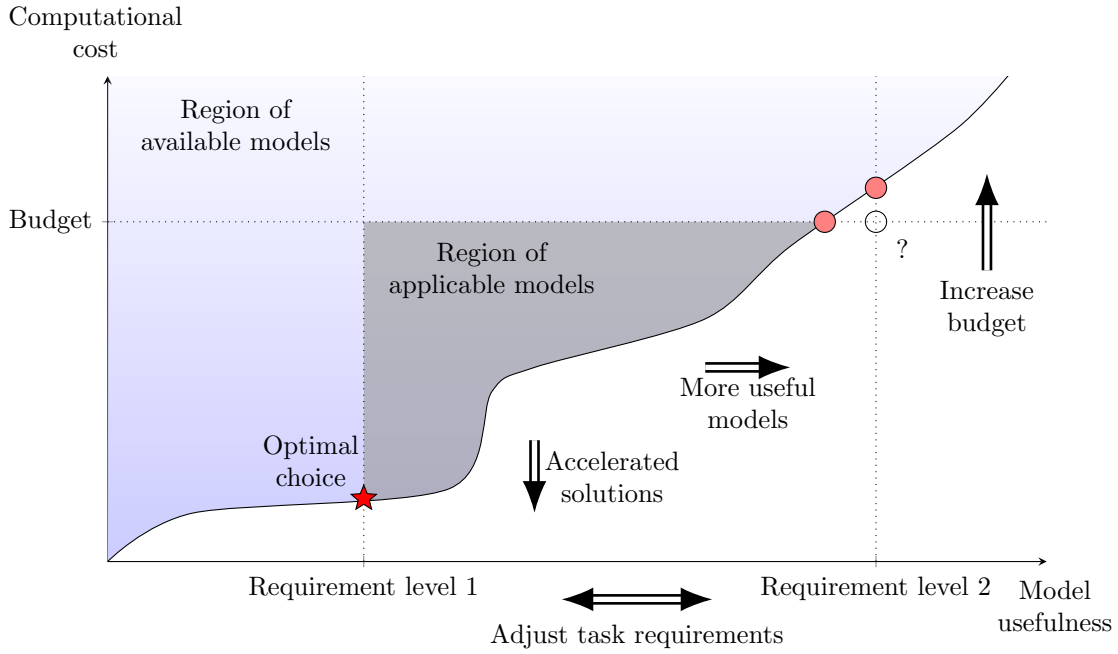


Figure 1.1: Trade-off between the usefulness of models (do they help to solve a task) and the computational cost of their solution. The minimum required usefulness and the computational budget limit the region of applicable models. The energy transition reduces this region, whereas more useful models and accelerated solutions extend the area.

it comes at increased financial needs for the infrastructure. Adjusting the requirements towards the model demands for accepting compromises with respect to operation. We could, for example, simplify very complex component models to lighten the computation at the risk of not being able to capture certain harmonics in the grid. From the modeller's view – which we will take – the region of applicable models needs to be extended in such situations. This can mean finding more useful modelling approaches as well as reducing the computational cost associated with a model's evaluation, i.e., accelerate the model solution process.

The energy transition towards a power grid with high shares of renewable energy sources challenges our modelling and operation approach [8]. In particular, more generation units, higher uncertainty, and faster responding components are a few of those challenges [7]. Trade-offs were always necessary, in particular with respect to real-time decisions, but they could be tested over many years. In contrast, the energy transition reshapes the power grid at a high pace and requires more difficult trade-offs. As a consumer of electricity, we usually only notice problems in the operation of the grid in case of a blackout. However, on a societal level, these trade-offs have very real implications: The amount of cheap and carbon-neutral electricity that is curtailed rises [9]. Reducing these inefficiencies is highly desirable from an economic and an environmental perspective and this can partially be achieved by better operation. Enabling grid operators to do so by enlarging the region of applicable models is the goal of this thesis.

1.2 Research gaps

Our focus lies on accelerating Time-Domain Simulations (TDSs) that are a key tool to analyse stability phenomena. When performing a TDS, we approximate how a *dynamical system*, such as

the power system, evolves over time. We can describe the instantaneous changes with differential equations in a very detailed and widely applicable form. However, solving a differential equation, i.e., tracing the system evolution over a certain amount of time, is extremely challenging. Even for seemingly simple differential equations, we often cannot solve them exactly but have to revert to numerical approximations [10]–[12].

The developed numerical methods are also applicable to dynamics encountered in the power system. However, there are several challenges that encouraged the development of tailored numerical methods. First, the high number of dynamical components, e.g., generators or inverters, leads to a large number of coupled non-linear Ordinary Differential Equations (ODEs). Second, the time-scales of interest can range from milliseconds to minutes depending on the chosen level of modelling detail. Third, the presence of algebraic and discrete variables requires special attention. Addressing these difficulties has been the subject of research over many years [3]–[5], [13]. The developed numerical methods are implemented in open-source, e.g., [14]–[16], and commercial software packages, e.g., [17], [18], and are the workhorse for many power system analyses. Recent research largely focuses on the potential of parallelisation, decomposition techniques, and Semi-Analytical Solution (SAS) methods [19]–[23]. These approaches primarily exploit the network structure in the power system to accelerate the solution process. This structure originates from the sparse network connections between the components which affects the interaction of their dynamic response.

Despite the recent improvements, the fundamental challenge of numerical methods for power system dynamics remains. It stems from the requirement to evaluate many (very) small time steps in order to provide results of acceptable accuracy. For a typical step size of 1 ms, the simulation of a dynamic response over a few seconds quickly entails thousands of time steps and even more internal function evaluations. The use of High-Perfomance Computing (HPC) clusters becomes a necessity for large grids as [24] highlights. Due to this computational burden, data-driven methods and Machine Learning (ML) have increasingly become a focal area of power system research [25]. Whereas the aim of these methods is often to circumvent TDSs for stability assessment, our goal will be to use ML to develop methods and tools to accelerate TDSs.

In that context, an entirely different approach to solving differential equations was proposed by Raissi, Perdikaris and Karniadakis in [26]. They utilise so-called Physics-Informed Neural Networks (PINNs) to *learn* the solution to a Partial Differential Equation (PDE), a form of differential equations often encountered in fluid dynamics. In the presented cases, the method eliminates the need for a time-stepping scheme and provides a continuous solution approximation; both are highly desirable solution characteristics. The authors in [27] adopt PINNs for the solution of a power system related ODE. Their results suggest a great potential for accelerating the solution of such ODEs while maintaining high accuracy. As the work in [27] considered the dynamics of a single machine, the question of the *scalability* of PINNs naturally arises. To be a direct alternative to established solvers for TDSs, i.e., to confirm the scalability of PINNs, it should be possible to simulate the dynamic response for a system with hundreds or thousands of dynamic components. The research gap, that we address in this work, is the need to understand if and how PINNs, as a modelling approach, can be utilised in the power system context for TDSs, i.e., understand the scalability of PINNs.

In fact, this research gap extends beyond the field of power systems. Before we can answer the question of the scalability of PINNs, we require a deeper understanding of solving a *small* system of ODEs with PINNs in the first place. PINNs have been applied to many problems involving

differential equations as [28]–[30] review. However, the test cases are predominantly governed by PDEs. ODEs, which are a special form of PDEs, have rarely been investigated as such. The mentioned reviews often treat them in unison as "ODE / PDE". This stands in stark contrast to the field of numerical analysis, where a distinction is made between ODEs and PDEs, see e.g., in [12]. Moreover, the study of non-linear dynamical systems and chaos, see e.g., [31], is primarily devoted to ODEs. By restricting the view to, only in some sense, the simpler form (ODEs) of differential equations, the development of problem-specific numerical methods is enabled. So-called symplectic solvers, for example, can ensure that the solution obeys conservation laws in dynamical systems. Including such insights and concepts in learning problems improves the outcomes as [32], [33] show. The cases where PINNs were applied to ODEs mostly concern the identification of system parameters. The topic of these studies are dynamics related to epidemiology, e.g., of COVID-19 [34]–[39], control problems [40], [41], and chemical kinetics [42]–[45]. The motivation for the last group of works originates in the stiffness of the ODEs that govern chemical kinetics. Stiff ODEs arise when the time-scales of the dynamics vary significantly and their solution requires specific solvers or solution schemes. Building understanding of how such characteristics influence the solution process when using PINNs, brings us back to the need to investigate PINNs specifically for ODEs in more detail. The alignment of this investigation with ODE-tailored numerical approaches could also be a key to enable scalability.

1.3 Research directions

Based on the research gaps identified in the previous section, we define two research directions (RD) that we will pursue in this thesis.

RD 1 PINNs as a modelling approach for dynamical systems

This RD focuses on the development of a thorough understanding of the methodological proposition that PINNs offer for solving ODEs. This includes placing the problem formulation in the context of dynamical systems that are governed by ODEs. We will use a simple dynamical system (a single machine) to ease the interpretability and the visualisation of the system and the learned solutions. The aim is to build a solid understanding that enables us to proceed to addressing the scalability of PINNs.

RD 2 PINNs for TDSs in multi-machine power systems

With this RD, we target the application of PINNs to TDSs for multi-machine systems. The methodological characteristics that determine the scalability need to be identified. Ultimately, the goal is extending the region of applicable models, i.e., a complementary use of PINNs and established (scalable) numerical solution schemes. By recognising their respective strengths, it shall be investigated how each solution method is best applied, potentially in conjunction in hybrid schemes.

1.4 Scientific contributions

This thesis pursues the goal of enabling the scalability of PINNs to be used for TDSs of power system dynamics. The acceleration of numerical methods for large-scale power system computations is an increasingly important enabler for an efficient energy transition. Adopting ML approaches to solve differential equations underlying these computations offers promising complementary (if not alternative) opportunities to the well established numerical methods.

Towards RD1, we provide a comprehensive investigation of PINNs for approximating the solution to a system of ODEs; this constitutes a *forward* problem. To the best of the author's knowledge, we are the first to introduce the framing of PINNs as *flow approximators* for dynamical systems. Instead of learning single trajectories of the dynamical system as in [27], we train PINNs for a range of initial conditions – we move from a trajectory approximation to a flow approximation. This view closely links to conventional numerics and approximation schemes. To separate the learning outcome from the choice of the ODE formulation and the scale of the dynamic states, we propose the use of a dimensionless state that incorporates the characteristics of the associated state space. Subsequently, we provide an overview of approaches to include domain knowledge of the dynamical system into the learning algorithm and propose two additional techniques. The first centres around an additional loss terms based on simulated data points – we call it the "dt-loss" and the resulting Neural Networks (NNs) "dtNNs". It penalises learned solutions that do not follow the governing ODEs around the simulated data points. The second extends implicit Runge-Kutta (IRK)-PINNs, as proposed in [26], from a discrete-in-time method to a continuous setting.

Furthermore, we consider the corresponding *inverse* problem, i.e., finding the parametrisation of the dynamical system based on data points. Drawing parallels from the statistics literature, we provide interpretations for and contrast the use of PINNs [26] and Bayesian Physics-Informed Neural Networks (BPINNs) [46] as Maximum Likelihood Estimators (MLEs) and Maximum a Posteriori Estimators (MAPEs) for the given parameter estimation task. Adopting a probabilistic viewpoint on the learning problem, enables us to provide an interpretation of the weighting factor α_c between the data-based loss and the physics-based loss, commonly used in the forward problem.

With respect to RD2, we demonstrate the numerical characteristics such as speed and accuracy on a 9, 11, and 39-bus system with 3, 4, and 10 synchronous machines, respectively. While being fast when evaluating the prediction, the required training of PINNs poses a substantial computational burden. We assess the overall computational cost by adopting the view from economics as variable (run-time or evaluation) and fixed (up-front or training) cost. To be a viable method from a computational cost perspective, a significantly large number of evaluations of the learned setup is required. We identify that the high dimensionality found in power systems poses a major barrier to broadly applying PINNs to TDSs with the current learning setup. To overcome this issue of scalability, we propose *PINNSim*, a novel algorithm for the simulation of power system dynamics. The algorithm utilises a separate PINN for each dynamic component and interleaves them with each other with a conventional root-finding algorithm. It allows benefiting from the advantages of PINNs found in RD1, i.e., providing fast and accurate flow approximations. We demonstrate that the proposed algorithm can achieve large time steps while being accurate and simple to implement. Thereby, PINNSim presents a modular solution approach that conceptually offers the potential for a scalable and fast simulator of power system dynamics.

Modelling in the context of power system dynamics needs to always consider being applied in a safety-critical environment; it becomes a question of "trust" into the modelling approach and the question naturally arises with ML-based methods. We, therefore, provide a reflection on the use of ML in the context of power system dynamics. As ML constitutes a very broad modelling approach, we suggest a classification of modelling approaches to better judge the expectations in terms of trust. Aligned with the illustration in Figure 1.1, the distinction becomes whether an ML-based approach aims at accelerating model solutions, identifying more useful models, or both at the same time. We describe challenges and opportunities of these approaches and the implications for trust.

1.5 Thesis organisation

The structure of this thesis follows the two research directions. Part I looks at PINNs from a methodological point of view with a focus on the ML setting and the analysis of PINNs as a numerical method for dynamical systems. Chapter 2 provides the modelling context by introducing important concepts and terminology. Based on these, we formulate the problem of solving ODEs and give a brief introduction into the relevant concepts of ML and numerics. In Chapter 3, we treat the forward problem, i.e., solving the governing ODEs. It focuses on what it means to learn such a solution, called *flow*, and how we can incorporate domain knowledge into the learning process. The aim of this chapter is to provide a deep understanding of the methodological proposition of PINNs as it forms the basis for the subsequent chapters. Chapter 4 then focuses on using PINNs and BPINNs for the inverse problem, i.e., finding a parametrisation of the ODEs that explains the observations. This chapter also adopts a probabilistic view on ML that proves helpful to understand ML as an inherently probabilistic method. In Chapters 3 and 4, we review the relevant literature at the end of each chapter to enable us to better compare PINNs with other methods.

Part II proceeds with the question of how PINNs can be used for TDSs of power system dynamics. After laying out the fundamentals of TDSs in power systems and reviewing current approaches in Chapter 5, we proceed with applying PINNs to multi-machine systems in Chapter 6. We identify the challenge of the "curse of dimensionality" and analyse the implications for learning-based methods. Following this analysis, in Chapter 7, we propose a hybrid solution approach named *PINNSim* that allows the integration of PINNs for individual machines into a conventional solution algorithm.

In an attempt to contribute with insights on the nascent research field of ML in power systems, Part III takes a step back to summarise learnings and reflections on the topic. Considering the safety-critical nature of power systems, large parts of the works in this thesis are motivated by the need to understand and potentially trust an ML-based approach to be employed in this context. To that end, Chapter 8 discusses some implications this need for "trust" in ML methods entails. We conclude in Chapter 9 and provide a brief outlook on future work.

The appendices provide additional information about the implementation and details about the training setup. Furthermore, we present additional results and provide the derivations for the formulations discussed in Chapter 4. An overview of the used notation can be found at the beginning of this thesis.

1.6 List of publications

Throughout the PhD project, the following publications have been prepared. For the purpose of a coherent presentation of the topic and the results, we do not include them explicitly in this thesis.

[**Pub. A**] J. Stiasny, S. Chatzivasileiadis and B. Zhang, 'Solving differential-algebraic equations in power systems dynamics with neural networks and spatial decomposition', (Submitted to 62nd IEEE Conference on Decision and Control (CDC)), 2023. DOI: 10.48550/ARXIV.2303.10256

[**Pub. B**] J. Stiasny and S. Chatzivasileiadis, 'Physics-informed neural networks for time-domain simulations: Accuracy, computational cost, and flexibility', (Submitted to Electric Power System Research (EPSR)), 2023. DOI: 10.48550/ARXIV.2303.08994

[**Pub. C**] S. Stock, J. Stiasny, D. Babazadeh, C. Becker and S. Chatzivasileiadis, 'Bayesian physics-informed neural networks for robust system identification of power systems', (Accepted at 2023 IEEE PowerTech), 2023. DOI: 10.48550/ARXIV.2212.11911

[**Pub. D**] J. Stiasny, S. Chevalier, R. Nellikkath, B. Sævarsson and S. Chatzivasileiadis, 'Closing the loop: A framework for trustworthy machine learning in power systems', in *Proceedings of the 11th Bulk Power Systems Dynamics and Control Symposium (IREP 2022), July 25-30, 2022, Banff, Canada*, 2022. DOI: 10.48550/ARXIV.2203.07505

[**Pub. E**] S. Chevalier, J. Stiasny and S. Chatzivasileiadis, 'Accelerating dynamical system simulations with contracting and physics-projected neural-newton solvers', in *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, ser. Proceedings of Machine Learning Research, vol. 168, 2022, pp. 803–816. [Online]. Available: https://proceedings.mlr.press/v168/chevalier22a.html

[**Pub. F**] S. Chatzivasileiadis, A. Venzke, J. Stiasny and G. Misyris, 'Machine learning in power systems: Is it time to trust it?', *IEEE Power and Energy Magazine*, vol. 20, no. 3, pp. 32–41, 2022. DOI: 10.1109/MPE.2022.3150810

[**Pub. G**] G. S. Misyris, J. Stiasny and S. Chatzivasileiadis, 'Capturing power system dynamics by physics-informed neural networks and optimization', in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 4418–4423. DOI: 10.1109/CDC45484.2021.9682779

[**Pub. H**] J. Stiasny, S. Chevalier and S. Chatzivasileiadis, 'Learning without data: Physics-informed neural networks for fast time-domain simulation', in *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 2021, pp. 438–443. DOI: 10.1109/SmartGridComm51999.2021.9631995

[**Pub. I**] J. Stiasny, G. S. Misyris and S. Chatzivasileiadis, 'Physics-informed neural networks for non-linear system identification for power system dynamics', in *2021 IEEE Madrid PowerTech*, 2021, pp. 1–6. DOI: 10.1109/PowerTech46648.2021.9495063

Part I

PINNs for dynamical systems

CHAPTER 2

Preliminaries: ODEs and ML

This thesis sits at the interface of three domains: Power systems, Machine Learning (ML), and numerics for Ordinary Differential Equations (ODEs). We will cover the power system aspect mostly in Part II. The methodological insights that we present in Part I, however, will be crucial to address the applicability and scalability of Physics-Informed Neural Networks (PINNs) to large-scale power systems. In the interest of a clear language throughout the remainder of this thesis, we will first provide an intuition of important concepts in the modelling context (Section 2.1). We then introduce the formulation of ODEs and numerical solution approaches (Section 2.2) before proceeding to a brief presentation of the relevant concepts from ML (Section 2.3).

2.1 Modelling – striving for generalisation

The following descriptions and the terminology are mainly based on [47]–[50]. They go into significantly more detail regarding these terms and provide more rigorous definitions. For the purpose of a concise description of ideas and observations in this thesis, the following concentrates on providing an intuitive understanding of fundamental modelling concepts. As this attempt will inevitably neglect subtleties that are present, we try to provide clarifications and references whenever the intuitive idea becomes imprecise.

Modelling can, on an abstract level, be seen as suggesting *models* that provide explanations for the relation between inputs X and outputs Y[1]. X and Y form the *experience* $\mathcal{E}$ on which we want to base the model and they can stem from observations or another model which then forms ground truth. This setup is illustrated in Figure 2.1. The model can state the necessary calculations for

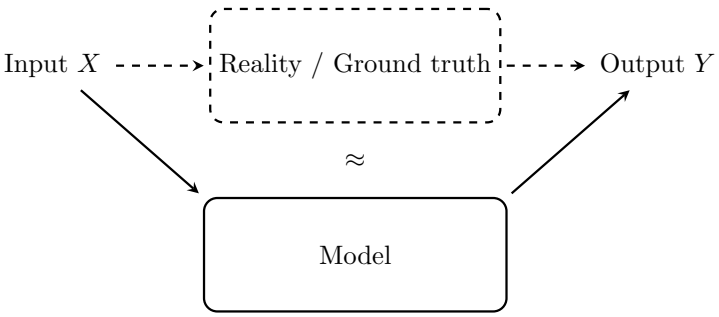


Figure 2.1: Illustration that views models as an approximation of the mapping between input X and output Y governed by reality or a ground truth model.

how to transform inputs X into outputs Y explicitly by a series of computations like additions, multiplications, etc., or implicitly by defining what relations the *mapping* between X and Y needs to satisfy. The *task* $\mathcal{T}$, which we try to solve, is to uncover how the relation between inputs and

[1]The notion of input and output implies a causality to determine which is which; such questions are left aside.

outputs can be described with the goal to understand and predict the present interactions. To assess how well a model expresses the mapping, we need to define a performance metric $\mathcal{P}$.

We assume that there is an optimal performance $\mathcal{P}^*$, which we consider being associated with the best model. To construct a model, we choose a hypothesis class h_θ that provides the structure of the model and by specifying the parameters θ we obtain a model, the best model uses the best parameter set θ^*. Potential structures of the model are polynomial functions, trigonometric functions, differential equations, Neural Networks (NNs), etc. Depending on the choice of parameters, a certain hypothesis class h_θ can result in different performances. This leads to a distribution of model performances for the different hypothesis classes as we illustrate in Figure 2.2. Some

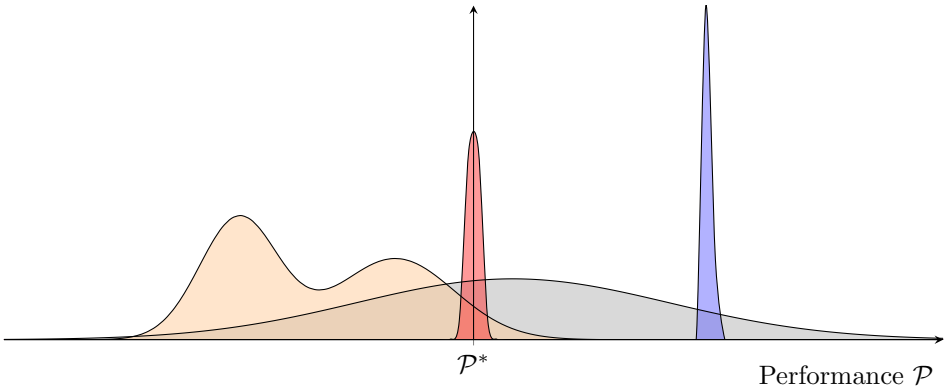


Figure 2.2: Distribution of model performances for different hypothesis classes h_θ. A narrow distribution corresponds to low variance (red and blue), the distance to the optimal performance $\mathcal{P}^*$ illustrates the model bias.

hypothesis classes will have very little variance in the distribution of the performance $\mathcal{P}$, given a credible choice of parameters. This is ideal, if the mean of the distribution centres around the optimal performance $\mathcal{P}^*$ as it is the case for the red distribution. However, it can happen that the variance is low but that we observe a significant bias (blue). On the other hand, there are hypothesis classes that lead to a distribution with high variance but low bias. For these models, parameter sets exist that yield a performance close to the optimal one, but other parameter sets perform badly (grey). These distributions are usually not unimodal, but much more complicated as hinted at by the orange distribution. The effect is that there can be a good proportion of reliably well performing models but also a significant amount of badly performing models.

What determines the kind of distribution a hypothesis class belongs to? This question leads us a few interconnected concepts relating to *representation capacity*, *model complexity*, and *generalisation*. For illustration purposes, we use the simple example of fitting a one-dimensional input variable X to a one-dimensional output variable Y. The example is inspired by [49]. As a note of caution: The explained concepts are much more intricate, hence, the provided intuition cannot replace detailed analyses when encountering more difficult modelling tasks.

We assume that we are given a number of data points, i.e., pairs of X and Y, shown in Figure 2.3. The task $\mathcal{T}$ is to find a model that can replicate these combinations of inputs X and outputs Y. A standard choice of a hypothesis class are polynomials $Y = \sum_{i=0}^{n} \theta_i X^i$ and one of the simplest models is assuming a linear relation between input and output, i.e., $n = 1$. When using the squared error between the model prediction $h_\theta(X)$ and the observed output Y as the model performance criterion $\mathcal{P}$, we can determine a set of parameters by minimising the squared errors between the

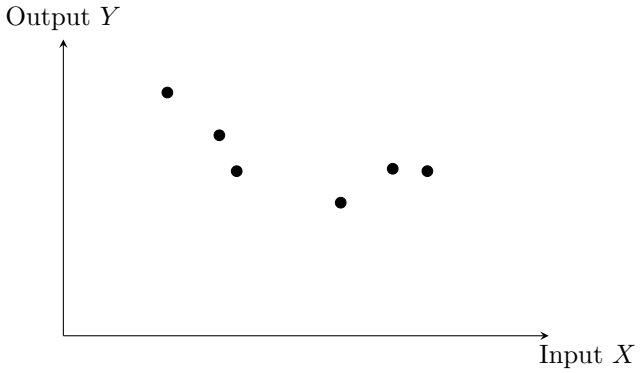
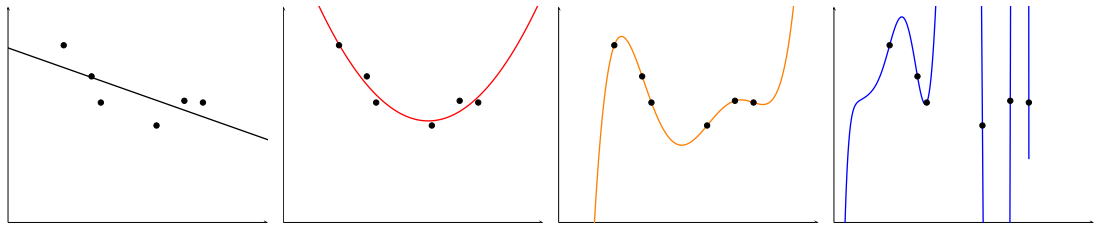


Figure 2.3: Dataset of observations with input X and output Y.

model and the data points. The result is shown in Figure 2.4(a). Similarly, we fit higher order polynomials ($n = 2, n = 5, n = 10$) also shown in Figure 2.4. Increasing the order of the polynomial



(a) linear h_θ $(n = 1)$ **(b)** quadratic h_θ $(n = 2)$ **(c)** h_θ with $n = 5$ **(d)** h_θ with $n = 10$

Figure 2.4: Polynomials of different orders fitted to the dataset in Figure 2.3.

increases the *representation capacity* of h_θ. This means that the higher-order polynomials can represent more complicated functions; however much we try fitting a linear h_θ, we cannot fit all points exactly – the linear model does not have sufficient representation capacity. When choosing $n = 5$ or $n = 10$, the polynomial can fit all points, i.e., the representation capacity is sufficient. It is important to note that the used higher order polynomials (hypothesis classes) could also be parametrised to resemble the linear hypothesis class by choosing $\theta_i = 0$ for $i > 1$. But as we optimised for $\mathcal{P}$ over the experience $\mathcal{E}$, i.e., the given data points, we arrived at a different parametrisation.

Which model should be chosen? To answer this question, we need new, unseen experience. From the given points, we could only conclude that we require $n \geq 5$ to exactly fit the data points. This brings us to the question of *generalisation*. In Figure 2.5, we plot the function (dashed line) from which the data points were generated, i.e., the ground truth. It is not difficult to tell, that with this additional information, the polynomial with $n = 5$ would be best suited, at least in the range with the white background; we say the model *generalises* well in this region. For the grey shaded areas, the predictions are not accurate. The models of lower order underfit the trajectory, the model with $n = 10$ is said to overfit. An underfitting model will have a bias in the distribution of the performances $\mathcal{P}$, it simply never can end up at $\mathcal{P}^*$. The overfitting model, in contrast, has enough representation capacity to yield models very close to the optimal value, but the performance can have an enormous variance. In the example, the 10-th order polynomial could simply use the same parameters as the 5-th order model while setting all others to 0, and perform much better than

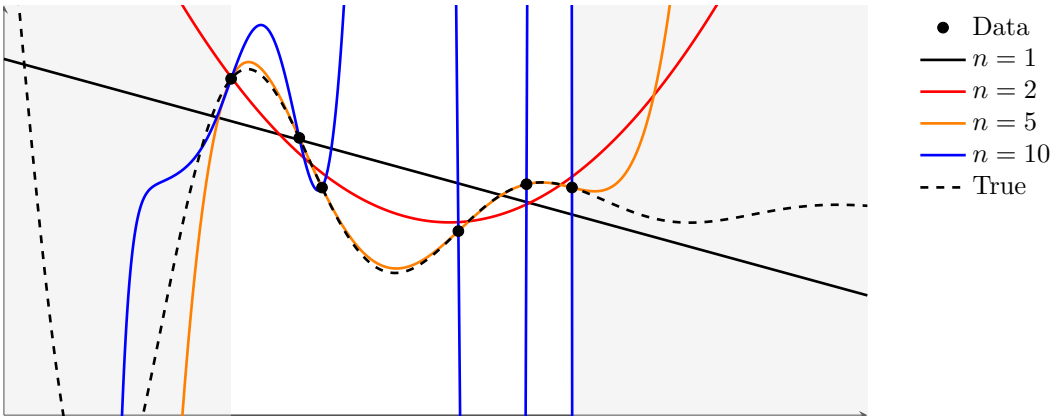


Figure 2.5: Comparison of the different fitted polynomial models of order $n = 1, 2, 5, 10$ with ground truth (dashed line).

with the shown parameters. Finding the balance between models with low variance but high bias and models with low bias and high variance is known as the *Bias Variance Trade-off (BVTO)* – this trade-off will accompany us through this thesis and the section on ML will come back to it. We can recommend consulting [48] for more details.

Shaping the BVTO to achieve good *generalisation* is the key of good modelling. Providing more data points is one way to do so, as it allows us to reduce the variance of the performance $\mathcal{P}$. The other way is to influence the *effective model complexity*. Without going into too great detail, the representation capacity defines an upper limit of what a model can represent. By using so-called *inductive biases* we aim to restrict this expressiveness to a level that leads to good generalisation. For the case of the 10-th order polynomial, we want it to "behave more like a 5-th order polynomial". In the thesis, we will often use the term *model complexity* instead of effective model complexity for brevity.

Inductive biases can take many forms as we will see later. As the name suggests, they *bias* the model, i.e., the performance distribution will change, and if designed well, the resulting bias is small whereas the reduction in variance is large. In all these considerations, we assumed a given performance metric $\mathcal{P}$, here, the squared error. When talking about generalisation, it is implicitly with respect to a certain performance metric $\mathcal{P}$. If we, for example, chose to measure model performance by the maximum absolute error, it can change the assessment of which models generalise well.

Coming back to the illustrative example, the true function is $Y = e^{-0.25X} \sin(0.875X)$. In fact, it is also the solution to the damped harmonic oscillator described by the ODE $\frac{d^2}{dX^2}Y + 0.5\frac{d}{dX}Y + Y = 0$ with the initial condition $Y(X = 0) = 0$. Here, we can very clearly see, that the number of parameters alone is not sufficient for talking about model complexity and representation capacity. By using trigonometric and exponential functions, the experience could be described in a model that generalises better than the polynomial hypothesis classes. Furthermore, choosing an entirely different hypothesis class h_θ, e.g., ODEs, can yield a model description that generalises well. The big difference is that ODEs define an implicit model that cannot directly be evaluated. The other examples were explicit models. A few ODEs have equivalent explicit models, i.e., their analytical solution (as in the example), but for the vast majority of ODEs this is not the case. The

attractiveness of ODEs persists nonetheless, as they allow the description of hypothesis classes h_θ that have a great ability to generalise.

The following section introduces the formulation (modelling) of ODEs and how to solve them. The NNs, that we introduce thereafter, constitute an explicit modelling approach, that is capable of representing extremely difficult mappings. However, bad generalisation is always looming.

2.2 Formulation and solution of ODEs

The problems, that we consider in Part I are defined by the change of the system state $\boldsymbol{x} \in \mathbb{R}^n$ with respect to time $t \in \mathbb{R}$ as a function $\boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$ of the state itself, i.e., by differential equations. The particular form where this change depends only on one variable renders it an Ordinary Differential Equation (ODE) and when time t is this variable, the system is usually considered a *dynamical system* [51] – Newtonian mechanics and the movement of objects are proto-typical examples of dynamical systems. The particular form, when $\boldsymbol{f}$ does not change with time constitutes an autonomous dynamical system. Whenever the system state $\boldsymbol{x}$ comprises more than one state, i.e., $n > 1$, we are dealing with a system of ODEs.

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}\left(\boldsymbol{x}(t)\right), \qquad \boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^n. \tag{2.1}$$

We generally assume that $\boldsymbol{f}$ is smoothly differentiable. Under this assumption, the motion that is defined by $\boldsymbol{f}$ will be unique for a given initial condition $\boldsymbol{x}_0 = \boldsymbol{x}(t_0)$. This constitutes a so-called initial value problem (IVP) and the Picard-Lindelöf-theorem assures the existence and uniqueness of the solution, we refer to [11] for a detailed description. To obtain the path of the motion – the *trajectory* $\boldsymbol{x}(t)$ – we integrate the changes over time

$$\boldsymbol{x}(t; \boldsymbol{x}_0) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}\left(\boldsymbol{x}(\tau)\right) d\tau. \tag{2.2}$$

The above trajectory describes the solution, given an initial state $\boldsymbol{x}_0$, as a function purely of time t

$$\boldsymbol{x}(t; \boldsymbol{x}_0) : \mathbb{R} \mapsto \mathbb{R}^n. \tag{2.3}$$

To solve the ODE for an arbitrary initial state $\boldsymbol{x}_0$, we need to describe a function that is known as the *flow* Φ [52]. By its defining property, it needs to satisfy

$$\frac{d}{dt}\Phi(\boldsymbol{x}, t) = \boldsymbol{f}\left(\Phi(\boldsymbol{x}, t)\right), \quad \Phi(\boldsymbol{x}, t) : \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n \tag{2.4}$$

$$\Phi(\boldsymbol{x}, 0) = \boldsymbol{x}. \tag{2.5}$$

If we evaluate the flow Φ for a particular value of time t, we obtain a so-called *flow map* $\phi_t : \mathbb{R}^n \mapsto \mathbb{R}^n$ defined as $\phi_t(x) = \Phi(x; t)$. An important property of flow maps is that their composition yields again a flow map. Hence, we can either apply a flow map with time step size h twice or the flow map for $2h$ once

$$\phi_{2h} = \phi_h \circ \phi_h. \tag{2.6}$$

This also entails that applying a flow map can be reversed by applying a time step of $-h$

$$\phi_0 = \phi_{-h} \circ \phi_h. \tag{2.7}$$

Solving the above equations analytically, that is finding an explicit expression for the flow Φ or a flow map ϕ_t, is only possible for a fraction of relevant cases. Exponential growth and harmonic oscillation are such cases. Usually, we are forced to rely on approximations. As we inevitably loose some key properties of flows and flow maps, we are interested in approximations that keep these inaccuracies in balance. This constitutes the key objective of designing numerical methods.

The approximation of the flow, of a flow map, or of a trajectory – we consider it the (inaccurate) *solution* – of ODEs solves the *forward* problem. When we observe instances of a flow or a trajectory and aim to reconstruct $\boldsymbol{f}$, we solve the *inverse* problem.

The following subsections present a brief overview over fundamental concepts in numerics, in particular of the Runge-Kutta (RK) schemes, for more detailed and comprehensive descriptions we refer to [12], [52]–[55].

2.2.1 Taylor series

A central idea to numerical methods is the approximation of a function by a series of other ("simpler") functions. The Taylor series is an important example, see, e.g., [11], [12]. It approximates a function $f(x)$ on the interval $[a, b]$ by an infinite series of polynomials. The coefficients of these polynomials are based on the n-th derivative $f^{(n)}(a)$ of the function f with respect to its input x evaluated at the point a (the 0-th derivative is simply the function itself)

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \tag{2.8}$$

In practice, we truncate this series, i.e., only consider terms up to order $n = r$; the truncated series is not equal to $f(x)$ any more, it becomes an approximation. Instead of using powers of $(x-a)$, i.e., polynomials, as the *basis* function, one can apply a similar idea but with different basis functions. The use of the trigonometric functions sin and cos, for example, leads to the Fourier series. As a consequence of the change of the basis functions, determining the coefficients requires a different procedure. The choice of the basis function is important, as the accuracy of the approximation will depend on it when only a finite number of terms r is used.

For functions operating on vectors, a similar expansion can be applied, but now we have to consider all interactions of the vector elements. It results in the need to compute the gradient, then the Jacobian, and then all higher order partial derivatives. This leads to a steep increase in the number of terms for higher order approximations. For this reason, the Taylor series, that we encounter in this thesis, often do not exceed order two.

2.2.2 Runge-Kutta (RK) schemes

The following introduces Runge-Kutta (RK) schemes, an important class of numerical integration schemes. The description in [12] provides an excellent introduction to this topic; the following summarises its key aspects that are of importance for this thesis.

When evaluating the integral (2.2) between t_0 and $t = t_0 + h$, where h represents a time step, we usually have the calculation of an area under the function in mind, know also as *quadrature*. An approximation of this area can be obtained by summing up small segments within the time step as depicted in Figure 2.6. This is known as a Riemann-sum. When we change the number of segments

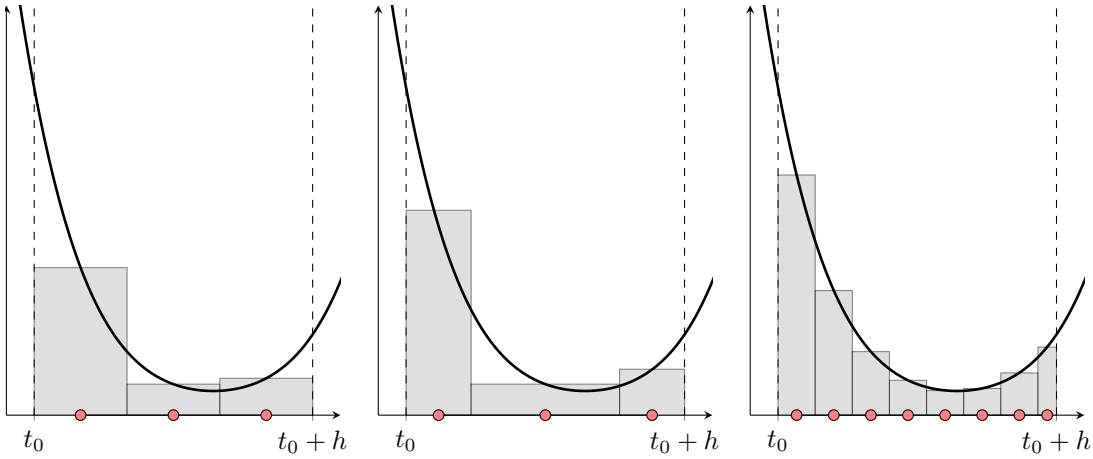


Figure 2.6: Riemann-sum for different number of segments (their location is indicated by the red dots) and different weightings, i.e., the rectangles' width.

and their placement within the interval, the value of the sum changes. By increasing the number of segments and calculating their sum, we can formulate a mathematical series. The limit value of this series defines the value of the integral, it is called a Riemann-integral. Efficient quadrature rules aim to use as few sections or function evaluations as possible for an accurate estimate of the value of the integral. This leads, for example, to trapezoidal integration or Simpson's rule. Gaussian quadrature describes the best approximation of the area given ν *nodes* by defining their location and relative weight.

Based on this idea of quadrature, we can approximate (2.2) by evaluating $\boldsymbol{f}$ at ν nodes that are placed at $t_0 + c_j$, where c_j is usually in $[0, h]$. The resulting values are weighted with b_j and summed up

$$\boldsymbol{x}(t_0 + h) \approx \boldsymbol{x}(t_0) + h \sum_{j=1}^{\nu} b_j \boldsymbol{f}\left(t_0 + c_j h, \boldsymbol{x}(t_0 + c_j h)\right). \tag{2.9}$$

However, to evaluate the above, we need to know values of $\boldsymbol{f}\left(\boldsymbol{x}(t_0 + c_j h)\right)$ at the nodes. By using an approximation $\boldsymbol{x}(t_0 + c_j h) \approx \boldsymbol{k}^j$ with $\boldsymbol{k}^j \in \mathbb{R}^n$ for each of the nodes, often called *stage*, we arrive at the so-called *Runge-Kutta (RK)* methods. They systematically describe how to obtain the stages $\boldsymbol{k}^j$ as a combination of other stages and the initial condition

$$\boldsymbol{k}^j = \boldsymbol{x}(t_0) + h \sum_{i=1}^{\nu} a_{j,i} \boldsymbol{f}\left(t_0 + c_i h, \boldsymbol{k}^i\right). \tag{2.10}$$

The parameters in (2.9) and (2.10) are summarised as

$$A = \begin{bmatrix} a_{1,1} & \dots & a_{1,\nu} \\ \vdots & \ddots & \vdots \\ a_{\nu,1} & \dots & a_{\nu,\nu} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_\nu \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_\nu \end{bmatrix}. \tag{2.11}$$

In principle, these parameters A, $\boldsymbol{b}$, $\boldsymbol{c}$ could be chosen independently, however, the resulting quadrature will not necessarily be accurate. In the following, we will describe the implications that the choice has on the characteristics of the resulting RK scheme.

Common RK schemes – explicit and implict

The major distinction of RK scheme can be made by how A is populated: If the matrix has a strictly lower-triangular shape (zeros on the diagonal and above), each stage can be explicitly evaluated. The first stage $\boldsymbol{k}^1$ equals the initial condition $\boldsymbol{x}(t_0)$, the second one $\boldsymbol{k}^2$ can only depend on $\boldsymbol{x}(t_0)$ and $\boldsymbol{k}^1$, and so on. These are *explicit Runge-Kutta (ERK)* schemes as each stage is formulated explicitly as a function of the previous stages. To predict $\boldsymbol{x}(t_0 + h)$, we evaluate (2.9) with $\boldsymbol{x}(t_0 + c_j h) \approx \boldsymbol{k}^j$. In contrast, *implicit Runge-Kutta (IRK)* schemes do not impose restrictions on A, but they result in a system of ν implicit equations. To obtain the values of the stages $\boldsymbol{k}^i$, we need to solve this system of implicit equations, which usually is non-linear. When solved, we can evaluate (2.9) in order to predict the state at the end of the time step.

The values of A, $\boldsymbol{b}$, $\boldsymbol{c}$ are often displayed in a so-called Butcher tableau, its structure is shown below to the left. Four relevant schemes, that we will employ are the Forward-Euler (FE) scheme, the Midpoint scheme, the 4th-order Runge-Kutta (RK4) scheme (often consider *the* RK scheme), and the Trapezoidal rule (the only IRK scheme of the four), their parameters are shown below.

Butcher tableau

$\boldsymbol{c}$	A
	$\boldsymbol{b}^\top$

Forward Euler

0	
	1

Midpoint

0		
$\frac{1}{2}$	$\frac{1}{2}$	
	0	1

RK4

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Trapezoidal

0	0	0
1	$\frac{1}{2}$	$\frac{1}{2}$
	$\frac{1}{2}$	$\frac{1}{2}$

Speed

For considering the evaluation speed of RK methods, we have to distinguish between explicit and implicit methods. The evaluation of ERK schemes depends mainly on the number of stages ν as this determines how often $\boldsymbol{f}$ is evaluated. Slight variations can occur in relation to how many zeros are present in the Butcher tableau. For IRK schemes, the evaluation time depends on how fast the system of implicit equations can be solved. Typical solution schemes are based on the Newton-Raphson algorithm and its adaptations to improve the computational speed, see, e.g., [12]. The significant cost of solving large systems of implicit equations explains why IRK schemes with many stages are rarely encountered.

Accuracy – local approximation error and consistency

The other important property is the local truncation error. By using Taylor series expansions for $\boldsymbol{k}^j$ and substituting them in Equation (2.9), a large expression as a function of the step size h can be formed. By collecting all terms of a common order of h, i.e., the terms with the same exponent, and comparing them with a Taylor series expansion of the left-hand side of (2.9), one can derive conditions for the parameters A, $\boldsymbol{b}$, $\boldsymbol{c}$ that lead to the RK scheme matching the Taylor series. When the coefficient on the left-hand and right-hand side of this expansion match for all terms $h^1, \ldots, h^p$ up to power p, the *local truncation error* will be of order $\mathcal{O}(h^{p+1})$. The scheme is said to be of *order* p. Up to $\nu = 4$, we can find parametrisations that yield $p = \nu$, for $p = 5$ we require $\nu = 6$, and it becomes increasingly difficult to achieve higher orders, i.e., we need larger

ν. For a subclass of IRK schemes, called *Gauss-Legendre methods*, one can prove that they are of order $p = 2\nu$, i.e., their errors are significantly smaller for the same number of stages compared to ERK schemes. This superior accuracy is traded against the increased evaluation cost due to the solution of the system of implicit equations. An important, even necessary, property for practical numerical scheme is *consistency*. It requires that the local truncation error approaches 0 when the time step size h approaches 0.

Usually, the time intervals, that we aim to evaluate, cannot be approximated in a single time step h due to insufficient accuracy. Instead, we apply a *time-stepping* procedure that splits the time interval into many small time steps that are in turn approximated with a RK scheme. It relates back to the possibility to compose flow maps. As each step has a local approximation error, these errors will compound and lead to a global approximation error at the end of the time interval.

Numerical stability

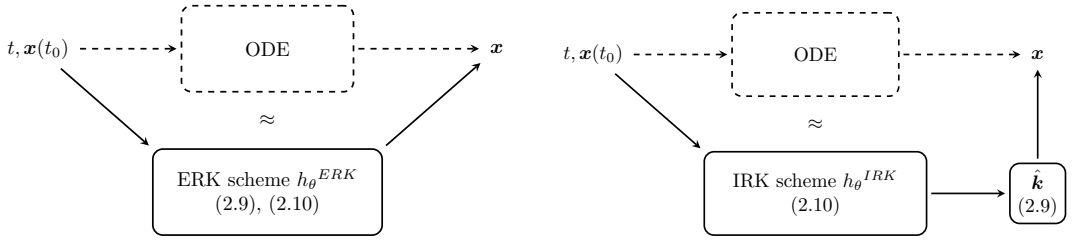
Numerical stability is a property, which becomes important when it is absent. This can occur for example for *stiff* ODEs. These ODEs contain state variations that settle at very different time-scales, e.g., within milliseconds and minutes. If the fast variations decay quickly and do not influence the slow variations (analytically), a numerical method should reproduce this behaviour. However, numerical instability arises from the fact, that any computation on a computer has round-off errors because we have limited precision, well explained in [12, p.54]. This error is usually not a concern when we evaluate functions once or twice. But for repeated evaluations, even an initially tiny round-off error *can* lead to an exponentially growing error – this is numerical instability. Whether this occurs or not, depends on the used numerical scheme and the ODE. IRK schemes can be designed to generally avoid this behaviour, i.e., the method is said to be *A-stable*, whereas ERK schemes can only rely on using small enough step sizes such that the error is "damped" sufficiently (this is not related to accuracy). The interested reader is referred to [12] and references therein for more details on stability, in particular when it comes to stability of numerical methods applied to non-linear systems. For this thesis, though, the existence of the issue of numerical (in)stability is of importance, as it is a common driver for the choice of an integration method.

2.2.3 RK methods as a model for the flow Φ of an ODE

We want to (very explicitly) highlight the following interpretation of RK schemes as a model for the flow Φ, i.e., for an ODE solution. This view is inspired by [52], where the author describes the goal of RK methods as producing a family of approximate flow mappings depending on h. Adopting this perspective will become useful when we compare numerical integration methods with PINNs as alternative model of the flow.

To this end, we adopt the generic modelling structure shown in Figure 2.1. When applied to ERK and IRK schemes, we obtain Figure 2.7. The inputs in the context of modelling ODEs are time t and the initial condition $\boldsymbol{x}_0$ and the output is the state $\boldsymbol{x}$ after a time step $h = t$. A RK scheme becomes a model for the relation between inputs and outputs that is described by an ODE. The hypothesis class h_θ for RK methods consists of (2.9) and (2.10) and the parameters $\boldsymbol{\theta}$ are $A, \boldsymbol{b}, \boldsymbol{c}$. In contrast to other modelling approach, we determine the parameters $\boldsymbol{\theta} = A, \boldsymbol{b}, \boldsymbol{c}$ not based on any experience $\mathcal{E}$, but directly based on the original ODE model.

The ability of this model to generalise can be analysed by the obtained accuracy of the particular RK method. Unlike many other modelling endeavours, we can describe generalisation well, as we



(a) ERK scheme **(b)** IRK scheme

Figure 2.7: Explicit and implicit RK schemes from a modelling perspective.

can express the order of the local truncation error as a function of h. Hence, the model's capacity to generalise depends directly on the time step size h but not directly on the initial condition and "only" insofar on the ODE formulation as numerical stability is concerned. Still, the exact generalisation error is unknown, as a statement like $\mathcal{O}(h^2)$ only describes the asymptotic error. This means that the error outside the asymptotic region is not captured and, furthermore, the magnitude is unknown. However, the great advantage is, that by reducing h, we are guaranteed to reduce the local approximation error if numerical consistency of the scheme is given. And, furthermore, error approximation schemes allow for a good control of the modelling error. These are the great assets of numerical schemes; we know that they do generalise extremely well when used in a suitable range. For IRK schemes, we furthermore have to add the solution of the system of implicit equations as an additional step in the modelling that can introduce errors (it could be viewed as another model itself). For solving ODEs, RK schemes can therefore be seen as approximation of the flow, we denote such approximations by $\hat{\Phi}$. The hat symbol will generally be the notation for approximations/estimates.

There is another class of numerical methods that is widely used: multistep methods. These methods utilise not only the current state, but incorporate previous states, i.e., the states that had to be computed beforehand. While this approach works very well in practice, strictly speaking, it constitutes a different modelling setup; the inputs to the hypothesis class h_θ of a multistep method are the past states along with the current state. For this reason, we do not further consider it here. We will use one type of multistep methods (a Backward Differentiation Formula (BDF)) in Part II as a baseline method.

2.3 Machine learning formulation

The generic ideas of modelling explained in Section 2.1 apply also to ML and our goal of the following paragraphs is to deliver a more precise definition of some terms introduced in Section 2.1.

In the context of this work, the task $\mathcal{T}$ is often the solution of ODEs, i.e., finding flow approximations $\hat{\Phi}$. The previous section introduced a selection of how to derive functions such as the RK-methods that solves the task $\mathcal{T}$ without the need for any experience $\mathcal{E}$, the definition of the ODEs is sufficient. The performance metric $\mathcal{P}$, that was implicitly aimed for, is the size of the error between the true and the approximated flows, a relation governed by the local truncation error. The ML-based methods in this thesis, i.e., PINN, are just another way of obtaining a candidate model that solves the task $\mathcal{T}$ – the performance metric $\mathcal{P}$ will decide whether they are attractive to use or not.

To specify the learning algorithm, we have to translate task $\mathcal{T}$, performance metric $\mathcal{P}$, and experience $\mathcal{E}$ into a form that is algorithmically solvable, we need to set up a *learner* [47]. The chosen form is the following optimisation problem

$$\min_{\theta} \mathcal{L}(\mathcal{D}; \boldsymbol{\theta}). \tag{2.12}$$

We transform the performance metric $\mathcal{P}$ into a so-called *loss* $\mathcal{L}$. The two can be identical, but more often than not this is a design choice that influences the optimisation difficulty. The second component of a learning algorithm is the *hypothesis class* h_θ with the adjustable parameters $\boldsymbol{\theta}$. By evaluating the loss function $\mathcal{L}$ on a dataset $\mathcal{D}$, in which experience $\mathcal{E}$ is contained, we obtain a single objective value. It is dependent on $\boldsymbol{\theta}$ and is subsequently optimised. Eventually, tThe trained hypothesis class h_θ can be used for solving the task $\mathcal{T}$.

2.3.1 Dataset

The experience $\mathcal{E}$ provided to the learning algorithm stems from a *data-generating process* that defines how the inputs X are related to the output Y. It can be split into a deterministic mapping g and a noise term ε, that is often assumed to be normally distributed $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$,

$$Y = g(X) + \varepsilon. \tag{2.13}$$

The dimensionality of the involves quantities will be problem dependent, therefore, we refrain from specifying the involved dimensions and assume one-dimensional variables for the remainder of this section. In subsequent chapters, the dimensions of the input and output variables will be specified.

When collecting a dataset $\mathcal{D}$ of N samples (indexed by the superscript (i))

$$\mathcal{D} : \{(X^{(1)}, Y^{(1)}), (X^{(2)}, Y^{(2)}), \dots, (X^{(N)}, Y^{(N)})\}, \tag{2.14}$$

we observe, how samples from a *data-generating distribution* $p_{\text{data}}(X)$ relate to the output Y. This constitutes the setting of a classic supervised learning problem as described in [48], [49].

For the most part of this thesis, we assume that $\sigma_\varepsilon = 0$, i.e., we operate in a noise-free setting, the data-generating process is fully deterministic. This holds whenever we treat forward problems and the only exception is Chapter 4, in which we solve an inverse problem. Furthermore, we do have access to the data-generating distribution $p_{\text{data}}(X)$, i.e., we can define it. As we also know the mapping $\boldsymbol{g}$, it will be defined by the flow Φ of the ODE, the generation of the dataset is entirely in our hands. This, for ML unusual, setting occurs, because we know the ODE model that forms the ground truth. In effect, we are aiming to replicate a data-generating process with a model, that is faster to evaluate than by using a conventional ODE solver.

2.3.2 Hypothesis class

The hypothesis class h_θ defines how we manipulate the input X to obtain predictions $\hat{Y} = h_\theta(X)$. It is parametrised by the parameters $\boldsymbol{\theta}$ which we adjust while learning. The hypothesis class can take many forms, it can be as simple as a linear transformation $\hat{Y} = A(\boldsymbol{\theta})X$ where the matrix $A(\boldsymbol{\theta})$ is to be learned, up to very complex NNs.

In this work, we focus on fully-connected feed-forward NNs, also known as multilayer perceptrons, hence any reference to NNs can be assumed to be of this type. The idea behind these NNs is,

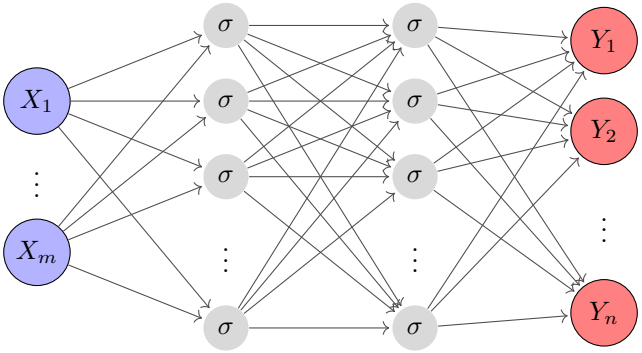


Figure 2.8: Architecture of a fully-connected feed-forward NN with two hidden layers.

first, to apply a linear transformation, defined by a weight matrix $W^{(k)}$ and a bias vector $\boldsymbol{b}^{(k)}$, and then a non-linear function $\sigma(\cdot)$. The combination of these two operations forms a *hidden layer*. By repeatedly applying such hidden layers, a NN can approximate very complicated functions. In fact, as proven early on by [56], [57], a NN with a just single hidden layer with certain non-linear functions constitutes a universal approximator, i.e., it can approximate any continuous function to arbitrary accuracy. In practice, however, such a NN would need to be very, very wide and finding the appropriate parameters θ would be impractical, as [49] elaborates. By using deep NNs, i.e., NNs with multiple layers, we nonetheless obtain a hypothesis class h_θ with a high representation capacity.

To formalise, the NN takes the form

$$\boldsymbol{z}^{(0)} = X \tag{2.15a}$$

$$\boldsymbol{z}^{(k+1)} = \sigma\left(W^{(k)}\boldsymbol{z}^{(k)} + \boldsymbol{b}^{(k)}\right) \quad k = 0, 1, ..., N_L - 1 \tag{2.15b}$$

$$h_\theta(X) = W^{(K)}\boldsymbol{z}^{(K)} + \boldsymbol{b}^{(K)} \tag{2.15c}$$

where the vectors $\boldsymbol{z}^{(k)}$, represent the intermediate output values of the hidden layers. We use N_L hidden layers and $\boldsymbol{z}^{(0)}$ simply constitutes the input. The weight matrices $W^{(k)}$ and bias vectors $\boldsymbol{b}^{(k)}$ form the adjustable parameters $\boldsymbol{\theta}$ of the NN. Their size depends on the width of the hidden layer, i.e., the number of neurons N_N. We will use the same width for all layers, hence $W^{(k)} \in \mathbb{R}^{N_N \times N_N}$ and $\boldsymbol{b}^{(k)} \in \mathbb{R}^{N_N \times 1}$. Exceptions are $W^{(0)} \in \mathbb{R}^{N_N \times \dim(X)}$, $W^{(N_L)} \in \mathbb{R}^{\dim(Y) \times N_N}$, and $\boldsymbol{b}^{(N_L)} \in \mathbb{R}^{\dim(Y) \times 1}$ to account for the size of the input and output variables X and Y. For the non-linearity $\sigma(\cdot)$, also called *activation function*, we use the hyperbolic tangent $\sigma(x) = \tanh(x)$, its output is bounded within $[-1; 1]$ and it is continuously differentiable. Other popular choices are the Rectified Linear Unit (ReLU) and the sigmoid function, see, e.g., [49] for more explanations. The output of the NN will be another linear transformation but without applying a activation function so that we can predict any real number. Figure 2.8 shows a schematic representation of such a NN (here with $X \in \mathbb{R}^m$ and $Y \in \mathbb{R}^n$).

This type of NN is a very generic and powerful hypothesis class h_θ, but it is also one of the most basic ones – we discuss the implications in Section 2.3.7. This has prompted the development of an entire zoo of hypothesis classes. Prominent ones are Convolutional NNs (CNNs)[58], Recurrent NNs (RNN) [59], and Graph NNs (GNNs) [60], we refer to [49] for a wider and more in-depth overview. Their development is driven by task specific properties that can be integrated in the hypothesis class. The use of the term *hypothesis class* h_θ helps, in the author's view, to abstract from a specific

NN architecture. It encapsulated the idea to define a model structure that determines how the trainable parameters shall influence the model's behaviour. It is analogous to using a polynomial function of a certain order, it defines h_θ, with a set of coefficients, the parameters $\boldsymbol{\theta}$.

2.3.3 Loss function

In order to *train* a NN, i.e., adjusting the parameters $\boldsymbol{\theta}$ of the chosen hypothesis class h_θ, we formulate an objective function that we can optimise. This *loss function* $\mathcal{L}$ should be chosen such that the optimisation is efficient and progressing well, while also giving a good performance $\mathcal{P}$ on the task $\mathcal{T}$ – the two are not necessarily the same. For regression problems, i.e., predicting a continuous value, we measure the difference between the prediction $h_\theta(X)$ and the target Y by a norm $\|\cdot\|$ and average it over all points in the dataset $\mathcal{D}$

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| Y^{(i)} - h_\theta(X^{(i)}) \right\| \tag{2.16}$$

A standard loss function is the mean-squared error (MSE) which we obtain when choosing the squared L^2-norm in (2.16). Other choices, such as the L^1-norm are possible, for regression tasks though, the L^2-norm is the common choice, we elaborate on this in Section 3.3.

One can modify this loss function by adding other terms. The most well-known examples of such regularisations are the ridge-regularisation and LASSO-regularisation which penalise large parameters and the non-zero parameter count, see, e.g., [48] for more details.

2.3.4 Optimisation algorithms

The optimisation of the loss function $\mathcal{L}$ poses a very high-dimensional and highly non-linear optimisation problem, given that even a simple NN can easily have thousands of parameters. To find well-performing parameter sets, a variety of optimisation algorithms have been developed based on an iterative procedure. In its simplest form, the parameters are iteratively updated using the gradient information $\nabla_\theta \mathcal{L} = \frac{\partial}{\partial \theta} \mathcal{L}$ of the loss function with respect to the parameters $\boldsymbol{\theta}$ scaled by the step size α

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha \nabla_\theta \mathcal{L}. \tag{2.17}$$

By repeatedly applying these gradient descent steps, we ideally converge, but generally we do not expect to find a global minimum. We refer to one update / iteration as an *epoch*.

The formulation of the gradient descent is based on a Taylor series expansion, here applied to the loss function $\mathcal{L}$. In principle, we can also use higher order derivatives to get a better local approximation of the loss function. This becomes prohibitively expensive though, once the number of parameters increases significantly as the size of the Hessian scales with $|\theta|^2$. However, even an approximation of the Hessian can be of substantial help for reducing the loss faster, i.e., learning faster. The Limited Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) optimiser (see, e.g., [49], [61] for a description), that we use in this work, is centred around this idea.

One of the most widely optimisers, named Adam [62], incorporates the curvature information of the loss function, i.e., the Hessian, by updating a *momentum* variable throughout the optimisation process. It is furthermore often applied to a subset of the dataset, which leads to a stochastic gradient descent (SGD) procedure.

2.3.5 Model selection and assessment

As described in [49, Ch. 8], *learning* differs from *pure optimisation*. The pure optimisation of the loss $\mathcal{L}$ over a training dataset $\mathcal{D}_{\text{Train}}$ does not necessarily lead to a good performance $\mathcal{P}$, the ultimate objective of the learning process. A hypothesis class h_θ with a high representational capacity, such as NNs with sufficiently many layers and neurons, could easily fit all data point in $\mathcal{D}_{\text{Train}}$, i.e., $\mathcal{L} = 0$. The prediction for an unseen data point, however, might have a high error. The expectation of this value is called the *test error* or *generalisation error*

$$\text{Err}_{\mathcal{D}} = \mathbb{E}[\mathcal{L}(Y, h_\theta(X))|\mathcal{D}_{\text{Train}}] \tag{2.18}$$

and it is conditional on the particular dataset $\mathcal{D}$ that the model was trained on. By formulation the expectation of this value over multiple datasets, we obtain an *expected prediction error* or *expected test error*

$$\text{Err} = \mathbb{E}[\mathcal{L}\left(Y, h_\theta(X)\right)] = \mathbb{E}[\text{Err}_{\mathcal{D}}]. \tag{2.19}$$

In the learning process, we estimate the generalisation error by calculating the loss on a separate dataset, the validation dataset $\mathcal{D}_{\text{Validation}}$. When this estimate, i.e., $\mathcal{L}\left(\mathcal{D}_{\text{Validation}}\right)$, begins to increase during training, it indicates the onset of overfitting. By stopping the training at the lowest value, we implicitly select the model that supposedly generalises best. This concept of *model selection* extends also to other parameters that influence the ability to generalise, such as the size of the NN or optimiser parameters; the term *hyperparameter tuning* is usually used in ML to refer to this selection of *hyperparameters*, i.e., the parameters apart from $\boldsymbol{\theta}$, that yield the best generalisation error. However, this process will introduce a bias as the model selection is conditional on the used validation dataset. To assess the model performance, i.e., conduct the *model assessment*, we need a separate test dataset $\mathcal{D}_{\text{Test}}$. It is important that the test dataset $\mathcal{D}_{\text{Test}}$ has not been used in the model training or selection to not compromise the estimate. For this reason, we perform a split of the dataset into training, validation, and test dataset, a typical split is, e.g., 60%/20%/20%. If the resulting datasets become too small, techniques such as cross-validation can be employed to improve the statistical significance of the estimates, we refer to [48] for details. Such techniques will not be employed in this thesis as we can simulate sufficiently large datasets, in particular for model assessment with the test dataset $\mathcal{D}_{\text{Test}}$. The statistical arguments that can be derived from this process of model selection and assessment assume that the data-generating distribution p_{data} is identical for all three datasets. We will see the implications of this assumption in Section 3.5.2.

2.3.6 Representation capacity and the Bias Variance Trade-off (BVTO)

Given the data-generating process $Y = g(X)+\varepsilon$ and using the squared-error loss, one can decompose the expected prediction error for an input $X = X_0$ (derivation as in [48, p.223])

$$\begin{aligned}
\text{Err}(X_0) &= \mathbb{E}[(Y - h_\theta(X_0))^2|X = X_0] && \text{(2.20a)}\\
&= \sigma_\varepsilon^2 + [\mathbb{E}[h_\theta(X_0)] - g(X_0)]^2 + \mathbb{E}[h_\theta(X_0) - \mathbb{E}[g(X_0)]]^2 && \text{(2.20b)}\\
&= \sigma_\varepsilon^2 + \text{Bias}^2\left(h_\theta(X_0)\right) + \text{Var}\left(h_\theta(X_0)\right) && \text{(2.20c)}\\
&= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance} && \text{(2.20d)}
\end{aligned}$$

Apart from Chapter 3, the irreducible error will be 0 as we learn in a noise-free setting. Hence, the error can be split into a bias-component and a variance-component. If we train models with

different hypothesis classes h_θ, the share of the bias and variance contribution will vary. When the representation capacity of the hypothesis class is low, the resulting error will be dominated by bias – recall the example in Section 2.1. If we fitted a different dataset in the same range of the input domain, the linear model will only change slightly; this variance in the models will reduce the more data points we sample. If, on the other hand, a hypothesis class has a high representation capacity, the error will be variance dominated; there is a wide range of 10-th order models that could fit the points. When we aim for a model that generalises well, we implicitly search for a model that has the optimal amount of representation capacity for a task. This idea is illustrated in Figure 2.9 and constitutes the BVTO. For a more detailed description, we recommend consulting [48], [49].

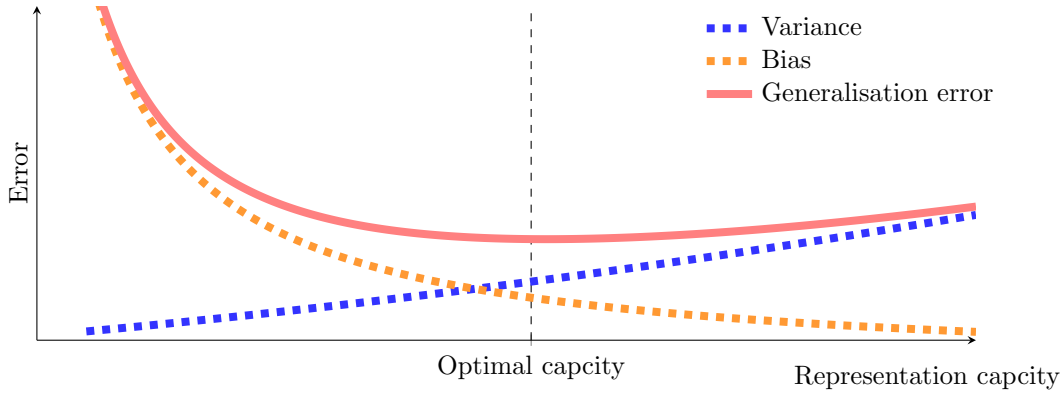


Figure 2.9: Schematic illustration of the Bias-Variance-Trade-off (BVTO), adapted from [49]

It should be mentioned that this view at the BVTO comes from the statistical learning perspective. The ever-increasing model complexity in deep learning would suggest that we should observe heavily overparametrised and hence overfitted models due to the variance. In practice, understanding generalisation appears to be much more difficult. We will not go into such learning-theoretical depth (the used models are anyway very shallow), but refer the interested reader to works on this topic [63], [64].

2.3.7 Inductive biases

The key to good generalisation in ML is introducing biases into the learning process that make it more likely that the learning algorithm arrives at a model that generalises well beyond the training dataset it has seen. This knowledge often originates from our experience – we know that rotating a picture usually does not change what is depicted – we get fooled or surprised when it actually does. The art of designing learning algorithms is to include such knowledge while not explicitly expressing every possible rule. From the problem formulation it is clear, that we can incorporate such domain knowledge in four ways: In 1) the hypothesis class h_θ, 2) the loss function $\mathcal{L}$, 3) the dataset $\mathcal{D}$ 4) the optimisation algorithm. The approaches can be summarised as designing *inductive biases*.

Ideally, we understand the effect of these biases, but there are also instances, where we observe a positive effect, i.e., a desired bias, without knowing exactly why it occurs. Regularisation is a well known inductive bias, that e.g., biases the model towards fewer parameters or smaller parameters. As mentioned before, the hypothesis class also has a great effect, Convolutional NNs (CNNs), for instance, help to obtain models that can deal with translation and rotation of pictures. Optimisers

are known to have inductive biases, e.g., stochastic-gradient descent, they are much less understood though, see, e.g., [65].

The idea of physics-informed ML, and in a broader sense of Scientific Machine Learning (SciML) (see, e.g., [28], [66]) is, that physical models and scientific computing methods provide a lot of knowledge that generalises very well, hence we should strive to incorporate it. The works in [26], [67] constitute the two landmark papers for the solution of differential equations. The term Physics-Informed Neural Network (PINN), termed by the authors in [26], is often used in the context of using differential equations in the loss function. However, the recent literature now investigates also hypothesis classes and dataset design in the context of learning the solutions to differential equations. Physics-guided, physics-inspired are other used terms that can be seen as applying inductive biases based on physical knowledge, or more specifically, information stemming from the differential equations.

2.3.8 Automatic differentiation (AD)

Many calculations in ML require the evaluation of gradients. The use of Automatic Differentiation (AD) for evaluating them has become a key enabler for training complex ML models. Modern ML frameworks such as PyTorch [68], TensorFlow [69], JAX [70], or Flux.jl [71], [72] implement AD as a central element. We refer to [73] for a detailed survey and explanation.

The fundamental idea is that a complex computation, such as evaluating a NN, can be expressed as a series of fundamental operations, e.g., additions, multiplications, or trigonometric functions. Their order and combination can be represented as a directed graph, the *computational graph*. In a *forward pass*, we start from the inputs and then step by step compute the elements in the graph. Importantly, we store not only the output but also *intermediate values*. By then reversing the direction of the graph, we can compute the derivate of the output(s) with respect to the input(s). For these computations we need to know the intermediate values and how to calculate the derivative of the fundamental operations. Essentially, it is the repeated application of the chain-rule. The basic implementation corresponds to *back-propagation*, however, by taking advantage of the graph structure, the calculations can be optimised in terms of speed and required memory storage. This is referred to as *reverse-mode AD*. For some calculations the *forward-mode AD* is beneficial, as the number of passes through the graph can vary. For the calculation of the loss in the forward pass, we need to evaluate the graph once. To then calculate the derivative of the loss with respect to the parameters $\boldsymbol{\theta}$, we only need a single reverse pass, which allows the scalability to learning problems with large number of parameters.

In contrast to numerical differentiation, AD is exact, as it does not require a discretisation interval. But it is also not symbolic differentiation, which allows the evaluation of the derivative without knowing the function value, i.e., no forward pass is necessary. Loosely speaking, with AD we evaluate an expression for the derivative that is locally exact.

CHAPTER 3

PINNs for forward problems

As motivated in Chapter 1, the power system can be described to great detail by large systems of Ordinary Differential Equations (ODEs). The used numerical methods are tailored to this type of differential equations and their particular structure. To achieve scalability of Physics-Informed Neural Networks (PINNs), it is of paramount importance to understand how this special functional form impacts the learning process and how it can be used to our advantage. As PINNs, so far, have been investigated pre-dominantly in the context of Partial Differential Equations (PDEs), we identify the need for an in-depth analysis of learning the solutions to ODEs. Therefore, this chapter centres around the learning task $\mathcal{T}$ of approximating the flow Φ of a dynamical system. The flow describes the solution to the governing ODEs for different initial conditions $\boldsymbol{x}_0$ and its approximation constitutes a solution to the *forward* problem. Flow approximators are central to the simulation of power system dynamics as they form the "atomic basis" on which multi-machine simulators are build. Hence, the insights from this chapter will be crucial for the goal of scaling PINNs to multi-machine systems in Part II.

In Section 3.1, we begin with an illustration of the dynamics that are associated with a Single-Machine Infinite-Bus (SMIB) system, which serves as the example of a dynamical system throughout this chapter. By comparing the PINN-based approximation of the flow Φ with other explicit numerical approximation schemes in Section 3.2, we identify the numerical characteristics that make PINNs attractive and further characteristics that are desirable. This leads us to the formulation of the learning task $\mathcal{T}$ in Section 3.3 and the initial "physics-agnostic" approach that utilises a vanilla Neural Network (NN) without any additional information on the governing dynamical system. In Section 3.5, we describe several ways of incorporating domain knowledge based on the governing ODEs, and illustrate their effects on the training outcome in Section 3.6. With these insights in mind, we elaborate in Section 3.7 on the characteristics of the learning task itself. To this end, we consider related approaches, potential learning setups, and the relation between ODEs and PDEs. We conclude and present a brief outlook in Section 3.8.

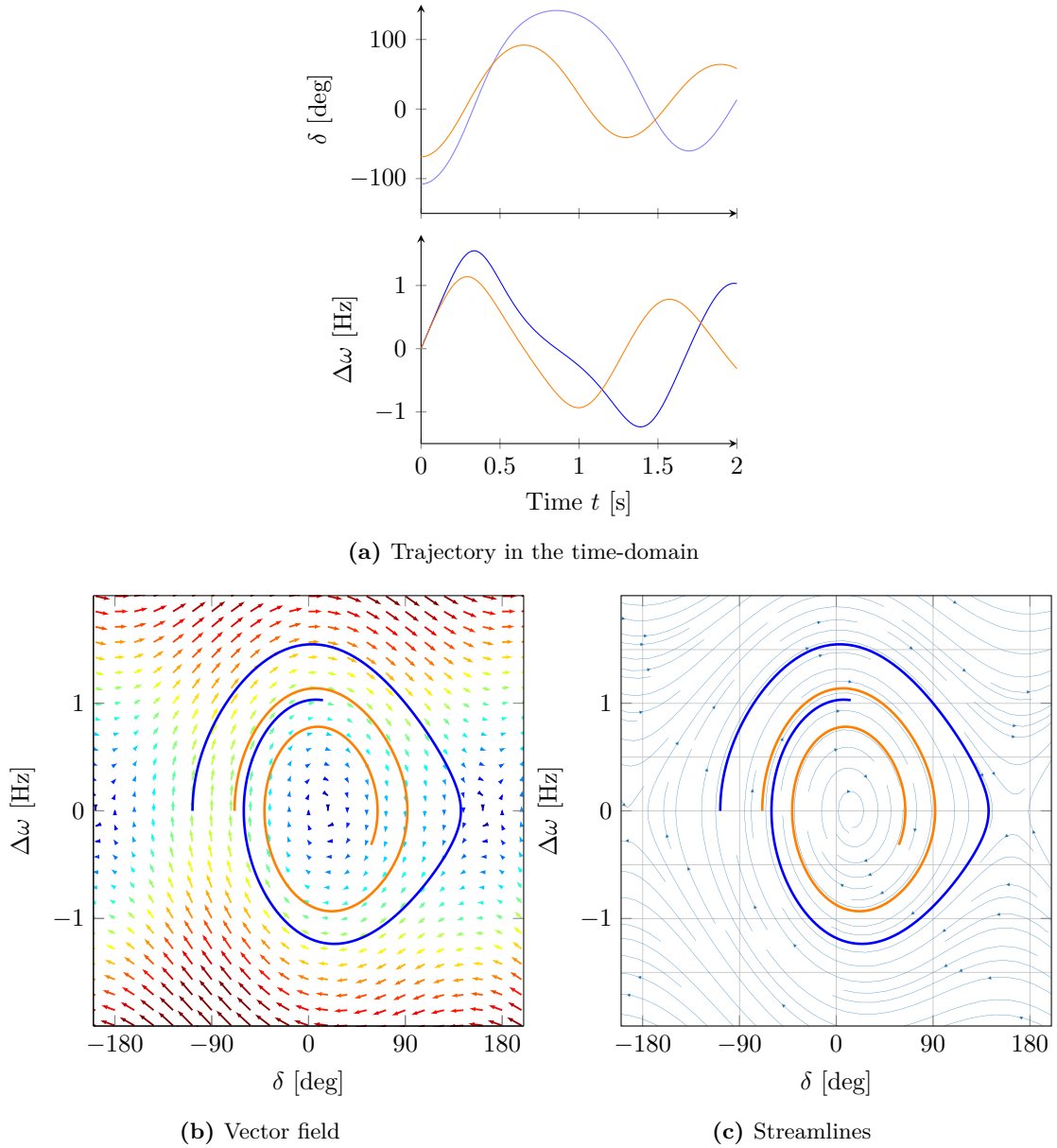
3.1 The flow Φ of a Single-Machine Infinite-Bus system (SMIB)

We begin this analysis by taking a closer look at the object of study, the Single-Machine Infinite-Bus (SMIB). In the power system context, it is a prominent setup for the dynamic analysis of components, and it usually serves as the starting point for more detailed analyses [3]. The underlying idea is to study the dynamics of a component that is connected to an infinite bus, i.e., a bus at which the voltage and the frequency are constant. For the component, we use a classical generator model that is described by the following ODEs

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\lambda}) \tag{3.1}$$

$$\frac{d}{dt}\begin{bmatrix}\delta \\ \Delta\omega\end{bmatrix} = \begin{bmatrix}\omega_0 \Delta\omega \\ \frac{1}{2H}\left(P_m - D\Delta\omega - \frac{E'_{q,0}V}{X'_d}\sin(\delta - \theta)\right)\end{bmatrix}. \tag{3.2}$$

At this point, we do not intend to dive into the physical details of the formulation, see Section 5.1 and [4] for a description and its derivation. Instead, we lay our focus on providing a more *geometric* understanding of the system. Not only will we use geometric interpretations throughout the chapter, this viewpoint also helps to see the SMIB system as an example of a dynamical system. In fact, (3.2) has the same structure as the ODEs associated with a non-linear pendulum, a classic example in dynamical system literature, see, e.g., [31], in this case with a damping and a forcing term. These dynamical systems consist of only two states, i.e., $\boldsymbol{x} \in \mathbb{R}^2$, hence it allows us to easily visualise the dynamics. For an n-dimensional state, i.e., $\boldsymbol{x} \in \mathbb{R}^n$, maintaining geometric intuition becomes difficult, if not impossible. For this reason, we use the state space formulation (3.1) for arguments and formulations, to the extent possible, and use the SMIB system only as an intelligible example.



(a) Trajectory in the time-domain

(b) Vector field

(c) Streamlines

Figure 3.1: Different forms of representation of the dynamics of the SMIB system. The orange and blue curves illustrate two examples of possible trajectories.

To understand a dynamical system, we can plot its temporal evolution as a trajectory $\boldsymbol{x}(t)$ as shown in Figure 3.1(a). The blue and orange trajectories originate from different initial conditions $\boldsymbol{x}_0$. Each state is considered separately, hence, it is a visualisation easily applicable to higher dimensional models. To understand the system characteristics though, the so-called *state space* (also called phase space) gives a much deeper understanding. The states, for the SMIB they are the rotor angle δ and the frequency deviation $\Delta\omega$, are plotted on the x and y axes. For each point in the state space, we can then evaluate the update function $\boldsymbol{f}(\boldsymbol{x})$, we assume $\boldsymbol{f}$ to be independent of time and with constant control input $\boldsymbol{u}$ and parameter settings $\boldsymbol{\lambda}$. By drawing these vectors and scaling them, we obtain the *vectorfield* in Figure 3.1(b). A trajectory can be constructed by taking small steps along the vectors of the vectorfield. Since there is a unique vector at each point in the state space, starting from a given state will lead to a unique trajectory[1]. We will subsequently use the streamline representation of this vectorfield shown in Figure 3.1(c), as it is a bit easier to follow. Each of the streamlines corresponds to a section of a trajectory. An important property of the here considered ODEs is, that the streamlines never cross. In the vectorfield and the streamline plot, time t is not directly visible. However, it is related to how fast we are moving along the trajectory. This "speed" can also be seen by considering the length of the vectors in the vectorfield.

The flow $\Phi(\boldsymbol{x}, t)$ of a dynamical system describes the evolution of states $\boldsymbol{x}$ with time. We can visualise it by tracing single points or a set of points as in Figure 3.2. Starting from $t = 0.0\,\mathrm{s}$, the points shown in (a) are propagated through the dynamical system. If we take snapshots of this process at times $t = 0.2\,\mathrm{s}$ and $t = 0.4\,\mathrm{s}$, we obtain plots (b) and (c). All three plots represent flow maps ϕ_t, namely $\phi_{0.0}$, $\phi_{0.25}$, $\phi_{0.5}$. In the following our goal is to find an approximation $\hat{\Phi}$ to the flow Φ which can also be viewed as a continuous set of flow map approximations $\hat{\phi}_t$ for ϕ_t.

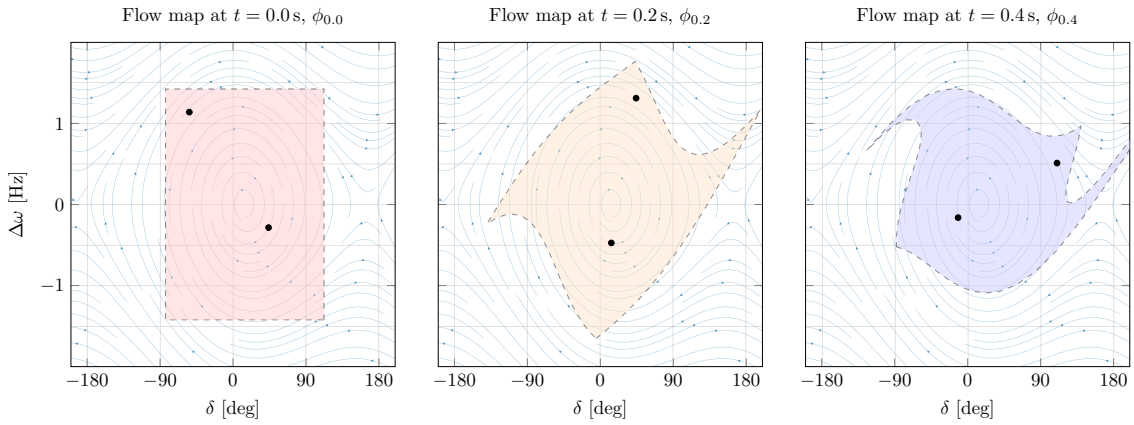


Figure 3.2: Flow maps ϕ_t of the SMIB system at $t = [0.0, 0.2, 0.4]$ s.

In Section 2.2, we introduced the Runge-Kutta (RK) schemes as a numerical integration scheme; the investigated PINNs constitute a class of functions comparable to these integration (flow approximation) schemes. Hence, the RK schemes will serve as important guidance for how to view and analyse the PINN-based flow approximation. We will cover other learning-based approaches for flow or flow map approximations in Section 3.7.

[1] This holds as we are dealing with an Initial Value Problem (IVP), well-defined under the Picard-Lindelöf-Theorem, see Section 2.2

3.2 Characteristics of flow approximators – PINNs and ERK schemes

When it comes to applying numerical flow approximators (numerical integration schemes), we are interested in two fundamental and critical characteristics: Properties related to their accuracy and properties of speed. Accuracy is usually analysed in terms of numerical aspects, such as the consistency, the stability and the convergence of a scheme[2]. The aspect of speed raises questions of algorithmic complexity, but also implementation, hardware, and programming language. We will discuss these topics in the following to demonstrate how PINNs perform in comparison to the explicit Runge-Kutta (ERK) schemes introduced in Section 2.2. These analyses will give an indication on desirable properties and we will show in the remainder of this chapter how we can exploit them in the application of PINNs or encourage them in the training process.

3.2.1 Approximation accuracy

We begin by showing the accuracy as a function of the time step size h and predictor type in Figure 3.3. The predictors are tested on a dataset stemming from simulations of the dynamics of the SMIB system. The solid line represents the maximum absolute error and the dashed line the median absolute error across this dataset. We consider explicit RK-based prediction schemes of different orders in the left panels. The use of higher order ERK schemes leads to a strong reduction of errors when the time step becomes smaller. The relation is well described by considering the local truncation error, in our case $\mathcal{O}(h^2), \mathcal{O}(h^3), \mathcal{O}(h^5)$, which decides upon the order of the RK-based scheme, i.e., a first, second, and fourth order scheme. As evident from the figure and local truncation error formulation, the error reduces when we reduce the time step size h, i.e., the schemes are numerically consistent. The order of the error holds for both states, however, their magnitude can be different. Their scale is implicitly given by the definition of the scale of the states and the update function.

The accuracy characteristics of PINNs are very different. We test two variants of PINNs, their construction will be the topic of the next sections. It illustrates that we have flexibility in PINNs to shape their characteristics as approximators. Their main advantage is that they provide, even for large time steps, accurate approximations. The accuracy only starts to deteriorate for points that were beyond the training domain indicated by the vertical black dashed line. On the other hand, for very small time step sizes the errors do not decrease (for the red version) or with $\mathcal{O}(h^2)$. We achieve the consistent behaviour of the latter at the expense of a slight increase of errors for larger time steps. If we compare the errors of RK and PINN schemes in absolute terms, we can identify ranges of advantages for one or the other. RK schemes are favourable for small h, then there is a region where PINNs perform better, and from around 1 s neither scheme gives accurate predictions. The region where PINNs can provide predictions with a near constant error while outperforming the RKs schemes, is the where the opportunity of PINNs lies. That being said, the first requirement is of course that the magnitude of the errors is acceptable for the application; if we, for instance, require a maximum error of less than 10^{-6} rad for the rotor angle δ, only the higher order RK schemes would be considered. We conclude, that there exists a "window of opportunity" for PINNs where they outperform the RK schemes and escape the usual relationship where increasing h leads to increasing errors. We will see that this characteristic can be shaped in the setup and training of PINNs (see Section 3.6.4) and also extends to larger dynamical systems (see Chapter 6).

[2]See Section 2.2 for consistency and numerical stability. The topic of convergence of numerical integration schemes for non-linear ODEs is beyond the scope of this thesis, we refer the interested reader to [11], [12].

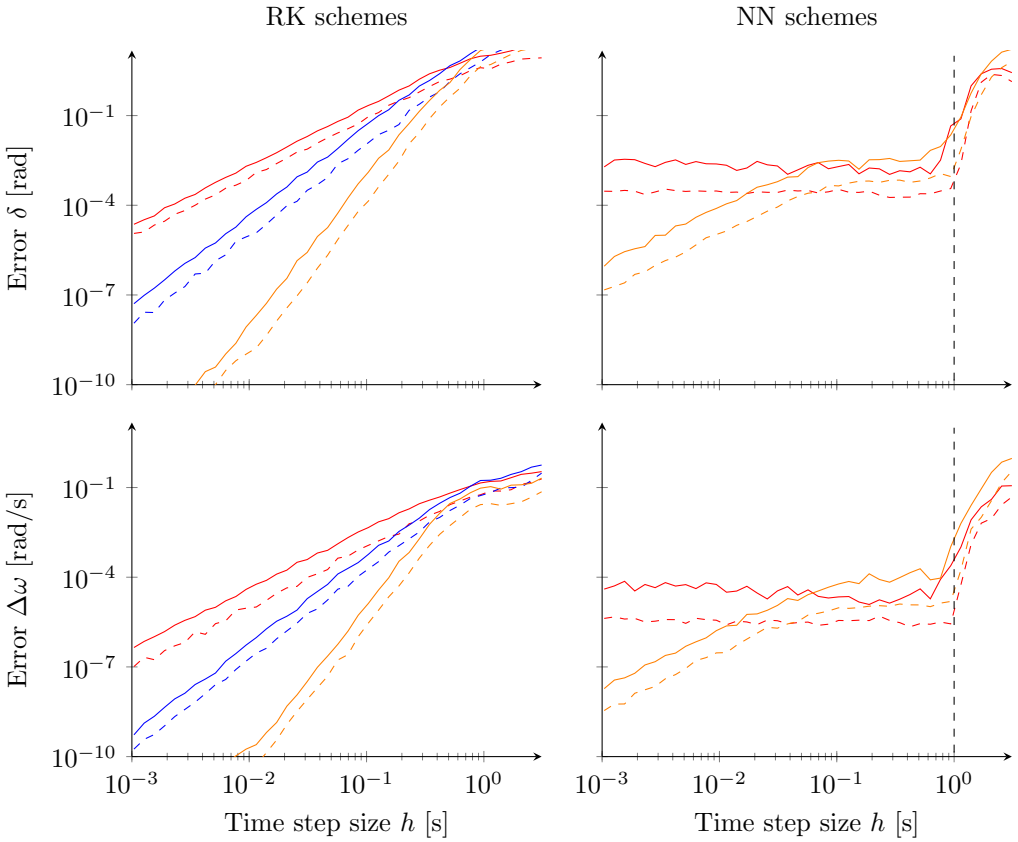


Figure 3.3: Approximation error as a function of the time step size h. The solid line represents the maximum absolute error and the dashed line the median error on the test dataset. In the panels to the left, we show a FE (red), Midpoint (blue), and RK4 (orange) scheme. In the right panels, we show a Residual (red) and Residual B (orange) architecture – their definition will be explain in Section 3.5. The black dashed line represent the maximum time step size in the training dataset.

3.2.2 Evaluation speed

The other important characteristic is evaluation speed. Obtaining reliable quantitative results of the evaluation speed is difficult as the efficiency of the implementation and the hardware play a significant role. In light of these dependencies, we first want to highlight that each of the above tested methods requires only a single evaluation, a *pass*, to obtain the approximation. This is the key characteristic of an explicit scheme; in an implicit scheme we need to solve the system of implicit equations first. The evaluation time of such a pass, denoted by C_P, then depends on the approximator used and this is best analysed by considering the computational time complexity. It describes how the computational time scales based on the most expensive operations. As for the analysis of the local truncation error, it does not specify an exact value but indicates the governing trend.

For RK schemes, $C_P = \mathcal{O}(\nu\, C_f)$ scales with the number of RK stages ν and the evaluation cost C_f of the update function $\boldsymbol{f}$. We do not go further into a detailed analysis of C_f as it is highly dependent on $\boldsymbol{f}$. The number of states $|\boldsymbol{x}|$ is an important factor but also the involved non-linearities and how many sequential operations we need to perform.

In contrast, the time complexity of PINNs is dependent on their architecture, in our case, the number of layers N_L and their size, i.e., the number of neurons N_N. For each layer we perform a

matrix-vector multiplication. If we assume all hidden layers to have N_N neurons, we obtain the complexity $C_P = \mathcal{O}\left((N_L - 1)N_N^2 + 2\,|\boldsymbol{x}|\,N_N\right)$. The first term corresponds to the evaluations of the hidden layers and the second term represents the impact of the input and output layer. The former will usually be the dominating one. This means that the evaluation time of PINNs is largely decoupled from the number of states $|\boldsymbol{x}|$ and entirely independent of $\boldsymbol{f}$ – attractive characteristics that clearly distinguish PINNs from ERK schemes.

To illustrate the difficulty of providing and comparing absolute numbers, we show in Table 3.1 a few examples for evaluation times. We distinguish by the used software, i.e., Python or Julia, by the used hardware, i.e., a Central Processing Unit (CPU) or a Graphics Processing Unit (GPU), and if we evaluate on a single or a thousand points. The reported time is the value per point for a function evaluation $\boldsymbol{f}$ C_f, for the computational cost of a narrow PINN ($N_N = 32$) and a wide PINN ($N_N = 256$), each with 4 hidden layers.

Table 3.1: Comparison of computation cost of a pass for the update function C_f, and for a narrow and wide PINN.

Software	Hardware	n points	C_f [s]	narrow PINN (4x32) [s]	wide PINN (4x256) [s]
Python	CPU	1	$986 \cdot 10^{-6}$	$702 \cdot 10^{-6}$	$1785 \cdot 10^{-6}$
		1000	$1.2 \cdot 10^{-6}$	$5.2 \cdot 10^{-6}$	$105 \cdot 10^{-6}$
Python	GPU	1	$1013 \cdot 10^{-6}$	$1262 \cdot 10^{-6}$	$1213 \cdot 10^{-6}$
		1000	$1.0 \cdot 10^{-6}$	$1.2 \cdot 10^{-6}$	$1.5 \cdot 10^{-6}$
Julia	CPU	1	$0.6 \cdot 10^{-6}$	$0.9 \cdot 10^{-6}$	$190 \cdot 10^{-6}$

A consistent trend is that the evaluation of 1000 points is significantly faster on a per-point-basis than for a single point. This is caused by overhead in the computations and the advantages due to vectorising operations. For the implementation in Julia, its pre-compilation allows a fast evaluation already for a single point. The hardware also plays a role, as seen for the difference between CPU and GPU. The latter is particularly suitable for matrix operations, hence, we see an enormous acceleration for the wide PINNs when using a GPU for the evaluation of many points.

We conclude that in a 4th-order Runge-Kutta (RK4) scheme, i.e., $\nu = 4$, the evaluation cost is comparable with that of the cost of evaluating the narrow NN, which makes the latter an attractive alternative evaluation speed-wise. However, in light of the wide range of results and the setup dependency, we decided to primarily use the metric of *passes* instead of absolute times to compare computational effort. This being said, the optimisation of the computational cost of each pass deserves an in-depth treatment in the future as the metric of evaluation speed hinges around it.

3.3 The task $\mathcal{T}$ of approximating the flow Φ

In the interest of a clear description of the methodology and results, we subsequently use the terminology introduced in Chapter 2. The formulation of the task $\mathcal{T}$ should be the first step of the modelling process. For the present case, the task $\mathcal{T}$ is the approximation of the flow Φ for the SMIB system with a hypothesis class h_θ that can be evaluated explicitly, i.e., in a single pass. We assess this approximation on a performance metric $\mathcal{P}$, e.g., the maximum absolute error. The

choice of this metric embodies what we care about. In numerics, bounding the maximum absolute error is usually crucial, aiming for a low absolute error on average might still be desirable, but it is secondary to the maximum error. We consider the task $\mathcal{T}$ *solved* when the model performance $\mathcal{P}$ meets a specified requirement, e.g., the error being less than 10^{-3}; specifying absolute or relative error tolerances is the terminology in numerics to set the requirements. The experience $\mathcal{E}$, based on which we learn, consists of a dataset $\mathcal{D}$ of samples from the true flow[3], i.e., without noise.

3.3.1 The domain of the task

In a first step, we restrict the input domain, on which we solve $\mathcal{T}$, to a subset of the state space $\boldsymbol{x}_0 \in \mathcal{X}_0 \subset \mathbb{R}^n$ and the time step size to the interval $h \in [0, h_{\max}]$. This means, for example, that we are only interested in the flow that originates from the rectangle in Figure 3.2 and for a time step size between 0 s and 1 s. We can query the model outside of this restricted domain, but the performance metric $\mathcal{P}$ will only depend on points within the input domain. This might seem restrictive but it is not all too different from RK schemes. When deriving them, we never specify the task $\mathcal{T}$ this formally. However, the performance of RK schemes is clearly dependent on the choice of the input domain in time; the analysis of the order of the local truncation error implies this. Therefore, we can only ensure the maximum error (a possible performance metric $\mathcal{P}$) to be less than a target value, e.g., 10^{-3} in Figure 3.3, by restricting the maximum allowable time step size h. Strictly speaking, a RK scheme, therefore. also only *solves* the task $\mathcal{T}$ on a restricted input domain. The advantage of a construction of the flow approximation like RK schemes is, that we usually do not need to specify the allowable initial condition[4]. This generalisability is difficult to achieve with a learning-based method, hence, we only aim to solve the task on the subset $\mathcal{X}_0$ in the state space.

3.3.2 The learning objective - from $\mathcal{P}$ to $\mathcal{L}$

What we learn is a design choice as we need to transform the performance metric $\mathcal{P}$ into an objective function of the learning algorithm, i.e., define the loss $\mathcal{L}$. In contrast to the other presented explicit methods, we have a great flexibility in shaping this transformation. At the same time, we also need to consider that the loss $\mathcal{L}$ will determine the optimisation landscape and this influences, along with the chosen optimiser, how "easily" we can learn a solution that performs well with respect to $\mathcal{P}$. Using the maximum error across the training dataset, for example, (L^∞-norm) is not a desirable choice for gradient descent-based optimisers. The gradient would depend only on the single point in the dataset that caused the maximum error. In the next iteration, another point could be responsible for the maximum error. This can lead to a very inefficient optimisation (it is of course problem dependent). Another potential choice is the L^1-norm, i.e., the sum of all absolute errors. The difficulty here arises from not factoring in the error magnitude. A prediction with many small errors can have the same loss as a perfect prediction for all but one point. When applying the performance metric $\mathcal{P}$ of the maximum absolute error, the former would clearly be the favoured outcome. This leads to the squared L^2-norm, i.e., the sum of the squared errors, a standard choice for regression problems. It assigns a greater importance to large errors while also

[3]When we refer to the "true" flow or solution of an ODE, we gloss over the fact that it still originates from an approximation-based integration scheme. However, this scheme is subject to very tight tolerances, e.g., $\varepsilon = 10^{-12}$ such that the occurring errors can be considered negligible.

[4]As the order of the local truncation error relates to the absolute error by an unknown factor, the initial condition can still have an impact on the fulfilment of the performance requirement. However, in practice, the adjustment of the time step size is used to meet the desired performance rather than restricting the allowable initial conditions.

having a favourable form for the optimisation. Furthermore, it closely links to a probabilistic view on learning as we will explore in Chapter 4. We will subsequently use this "natural" choice of the squared errors for the computation of $\mathcal{L}$, but want to stress that other choices are possible.

When the loss $\mathcal{L}$ is based on a single variable, a squared error is straightforward to calculate. However, for the case of approximating the flow of the SMIB system, we want to obtain a good approximation of both states, i.e., δ and $\Delta\omega$, at the same time. More generally speaking, as soon as the state variable $\boldsymbol{x}$ contains more than one state, we are faced with a multi-objective optimisation problem

$$\min_{\boldsymbol{\theta}} \sum_{j=1}^{\dim(\boldsymbol{x})} \xi_j \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} (x_j^{(i)} - \hat{x}_j^{(i)}(\boldsymbol{\theta}))^2 \tag{3.3}$$

that shall minimise the squared difference between the approximation $\hat{x}_j$ of the j-th state and the target value x_j. The scaling factors ξ_j determine how accurate the approximation of each state shall be; their choice is critical for the resulting approximation.

To illustrate this effect, we learn PINN models for different values of ξ_j, in fact, what matters is the ratio between the scaling of the two states $\xi_{\Delta\omega}/\xi_\delta$. Figure 3.4 shows the resulting maximum absolute errors across a collection of 20 PINNs trained with different initialisation seeds[5]. Importantly, we show the maximum absolute errors, i.e., the performance metric $\mathcal{P}$ for each state, not the loss value $\mathcal{L}$. In the trend, a small ratio reduces the error in δ while increasing the error in $\Delta\omega$, and

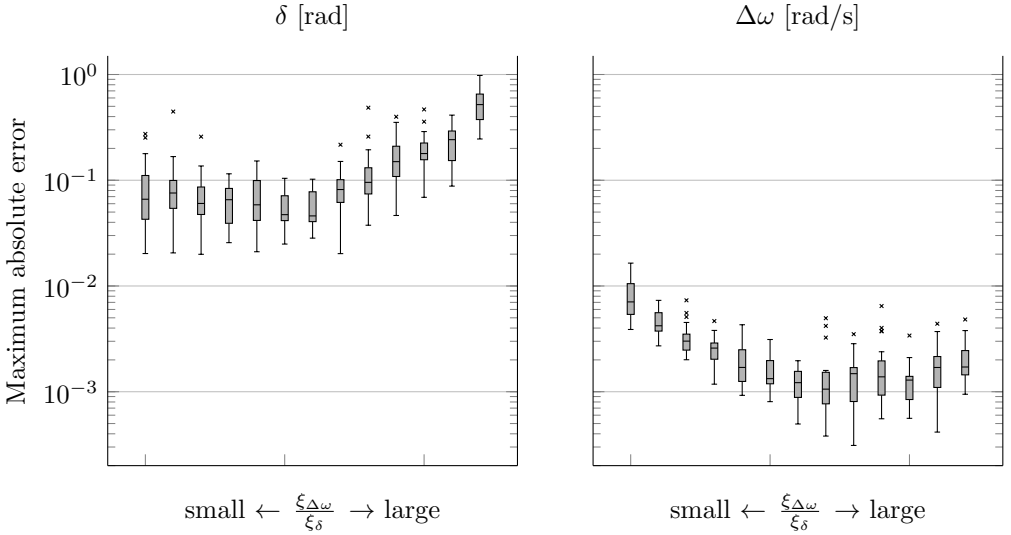


Figure 3.4: Impact of the ratio of the scaling factors $\xi_{\Delta\omega}/\xi_\delta$ on the maximum absolute errors of δ and $\Delta\omega$. The boxplots are calculated on the basis of 20 random initialisations.

vice-versa for an increasing ratio. The resulting errors can vary by an order of magnitude which underlines the importance of choosing an appropriate ratio. As it is also visible from the figure, the scales of the states are very different; we can observe the same in Figure 3.3. This has no "meaning" for the approximation schemes, however, in an application a prediction error of 0.01 rad in δ has a markedly different significance than an error of $0.01\,\mathrm{rad\,s^{-1}}$ in $\Delta\omega$. Bringing in the engineering perspective, we know appropriate scales of the states. For methods like RK schemes,

[5]To avoid the influence of other effects, we used here a vanilla NN. Therefore, the errors are larger than for the PINNs shown in Figure 3.3

we employ relative tolerances to account for these different scales when assessing errors. The choice of the scaling factors ξ_j serves the same purpose, but we have to include this choice directly in the learning formulation rather than applying it after training. Being oblivious to these aspects will lead to unsatisfactory learned solutions. This raises the question how to choose the scaling, which we discuss in the following.

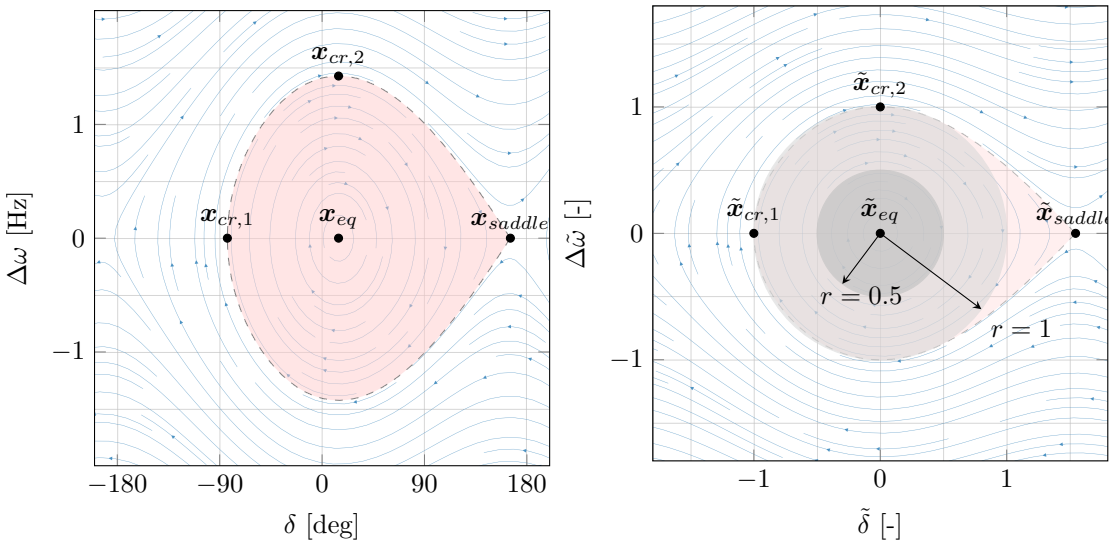
3.3.3 Learning in a dimensionless state space

The following aims at obtaining a more generic and adaptable approach to the selection of scaling factors ξ_i that ensures that the learning problem aligns with the engineering perspective. A very helpful guideline is whether we retain a meaningful understanding of the involved quantities. To this end, we propose the use of a dimensionless state $\tilde{\boldsymbol{x}}$

$$\tilde{\boldsymbol{x}} = \boldsymbol{T}(\boldsymbol{x}), \quad \boldsymbol{T} : \mathbb{R}^n \mapsto \mathbb{R}^n \tag{3.4}$$

for the formulation of the learning problem. Ideally, the transformation can be derived entirely based on the ODEs, i.e., the dynamical system's update function $\boldsymbol{f}$. This view is inspired by the idea of dimensionless quantities that are used, e.g., in fluid dynamics[6], for an introduction, see [74].

For the present case of learning the flow of the SMIB system, finding an appropriate scaling of the states is the prime objective. To that end, we use the geometry of the state space to inform the choice of $\boldsymbol{T}(\boldsymbol{x})$. For the undamped system, i.e., $D = 0$, Figure 3.5(a) shows the corresponding state space. Characteristic points in this plot (and of the dynamical system) are the stable equilibrium



(a) Original state space **(b)** Dimensionless state space

Figure 3.5: Transformation between the original and the dimensionless state space.

points $\boldsymbol{x}_{eq}$ and the saddle-node $\boldsymbol{x}_{saddle}$, their values can be obtained by solving $\boldsymbol{f} = \boldsymbol{0}$ for $\boldsymbol{x}$. If we followed a trajectory that originates very close to the saddle-node $\boldsymbol{x}_{saddle}$, we would obtain the dashed grey line in Figure 3.5(a). In the power systems context this defines the critical value V_{cr}

[6] In fluid dynamics, the rationale is that the dimensionless system description allows the characterisation of phenomena based on dimensionless quantities such as the Reynolds number. A model of a wind turbine in a wind tunnel can produce insights for the full-scale model if their Reynolds numbers (and other quantities) are matched.

of the energy function $V(\boldsymbol{x})$, that is commonly used in stability analyses, see, e.g., [3], [75]. For system states that lie inside the red shaded region, the system remains stable, for values outside it is considered unstable. Two additional characteristic "critical" points are $\boldsymbol{x}_{cr,1}$ and $\boldsymbol{x}_{cr,2}$ that mark the extreme values of δ and $\Delta\omega$.

From the learning perspective, an error of, e.g., 0.01, in either state should have a similar effect. An effect that we care about is whether the system is stable or not, i.e., are we inside or outside the region bounded by the critical energy level (the red shaded area). Based on this consideration, we choose the stable equilibrium point $\boldsymbol{x}_{eq}$ as the origin for the dimensionless state space. The points of the minimum rotor angle $\boldsymbol{x}_{cr,1}$ and the maximum frequency deviation $\boldsymbol{x}_{cr,2}$ form the unit vectors. Thereby, we obtain the transformation $\tilde{\boldsymbol{x}} = \boldsymbol{T}(\boldsymbol{x})$

$$\begin{bmatrix} \tilde{\delta} \\ \Delta\tilde{\omega} \end{bmatrix} = \begin{bmatrix} \frac{1}{\delta_{cr,1}-\delta_{eq}} & 0 \\ 0 & \frac{1}{\Delta\omega_{cr,2}} \end{bmatrix} \left(\begin{bmatrix} \delta \\ \Delta\omega \end{bmatrix} - \begin{bmatrix} \delta_{eq} \\ 0 \end{bmatrix} \right). \tag{3.5}$$

This transformation is only well defined for $\delta_{eq} < \frac{\pi}{2}$. Otherwise, we obtain a singular matrix and also physically, the stable equilibrium point is non-existent for $\delta_{eq} > \frac{\pi}{2}$. The inverse transformation $\boldsymbol{x} = \boldsymbol{T}^{-1}(\tilde{\boldsymbol{x}})$ is

$$\begin{bmatrix} \delta \\ \Delta\omega \end{bmatrix} = \begin{bmatrix} \delta_{eq} \\ 0 \end{bmatrix} + \begin{bmatrix} (\delta_{cr,1} - \delta_{eq}) & 0 \\ 0 & \Delta\omega_{cr,2} \end{bmatrix} \begin{bmatrix} \tilde{\delta} \\ \Delta\tilde{\omega} \end{bmatrix}. \tag{3.6}$$

The derivation assumed the undamped system, nonetheless, we apply this transformation also to the damped case, as the state space plots do not differ much in terms of the involved scales.

Of course, other choices of $\boldsymbol{T}$ are possible, however, the relation to the critical energy can potentially be generalised to other settings. Integrating Lyapunov theory and testing this concept on higher-order models will be future work. Furthermore, the investigation of a "full" dimensionless formulation, that also includes time t and reduces the system parameters to the minimal number, could become useful to generalise the learning across different values of $\boldsymbol{u}$ and $\boldsymbol{\lambda}$, as they determine the characteristics of the dynamical system.

Based on the defined transformation $\tilde{\boldsymbol{x}} = \boldsymbol{T}(\boldsymbol{x})$, we formulate a dimensionless loss $\mathcal{L}_x$ where all terms are weighted equally

$$\mathcal{L}_x = \sum_{j=1}^{|\boldsymbol{x}|} \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left(\boldsymbol{T}(\boldsymbol{x}^{(i)})_j - \boldsymbol{T}(\hat{\boldsymbol{x}}^{(i)})_j \right)^2 \tag{3.7a}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \tilde{\boldsymbol{x}}^{(i)} - \hat{\tilde{\boldsymbol{x}}}^{(i)} \right\|_2^2 \tag{3.7b}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)} \right\|_{T,2}^2 \tag{3.7c}$$

To avoid clutter in the notation, we will prefer to use (3.7c), where the subscript T indicates that the norm is to be calculated in the dimensionless coordinates. A further useful quantity will be the L^2-norm of a state in the dimensionless state space

$$r(\boldsymbol{x}) = \|\tilde{\boldsymbol{x}}\|_2 = \|\boldsymbol{x}\|_{T,2} \tag{3.8}$$

which we will refer to as the *radius* r. Figure 3.5 shows circles with $r = 1$ and $r = 0.5$. It is important to note that due to the non-linearity of the system, this radius does not stay constant along a trajectory or align with the levels of the energy function[7].

[7] A state space with concentric streamlines corresponds to an undamped harmonic oscillator that we can solve analytically.

One might argue that a standardisation of the data achieves the same effect. While this is true for the present choice, $\boldsymbol{T}(\boldsymbol{x})$ is not restricted to a standardisation and could also include non-linearities. More importantly though, if we need to simulate a dataset $\mathcal{D}$, we need to define the appropriate range of initial conditions, i.e., $\mathcal{X}_0$ from the previous section, in advance. As we will see in Section 3.5, this choice already requires the identification of the relevant region of the state space; we essentially need to identify $\boldsymbol{T}(\boldsymbol{x})$.

To summarise, the definition of the transformation $\tilde{\boldsymbol{x}} = \boldsymbol{T}(\boldsymbol{x})$ enables us to formulate a loss function $\mathcal{L}_x$ that aligns with the performance metric $\mathcal{P}$ in terms of the error scales. It allows us to shape the relative errors of the different states, however, they cannot be set independently but are part of a trade-off. This trade-off is a crucial difference to setting relative error tolerances in conventional numerical schemes as they can be chosen independently of each other.

3.4 The physics-agnostic approach to learning $\hat{\Phi}$

In this section, we illustrate a first attempt at learning a flow approximation $\hat{\Phi}$ with a vanilla NN without any further knowledge of the underlying ODEs – we call it "physics-agnostic" learning. The resulting approximator will be fast and under certain conditions also sufficiently accurate – these were the attractive approximation characteristics described in Section 3.2. These conditions relate to the fundamental concepts of modelling explained in Chapter 2: Generalisation, representation capacity, and the Bias Variance Trade-off (BVTO). By exploring these aspects in a physics-agnostic setting first, i.e., taking in no information from the physical system, it becomes clearer for the subsequent section, how "physics-informed" choices can contribute to meet the conditions for obtaining an accurate approximator.

Task and learning specifications

The task $\mathcal{T}$ will be the approximation of the flow of the SMIB system on the domain

$$h \in [0, 1] \tag{3.9}$$

$$r(\boldsymbol{x}_0) \leq 1. \tag{3.10}$$

The hypothesis class h_θ will be a vanilla NN as defined in (2.15) with the input $\boldsymbol{z}^{(0)} = [h, \boldsymbol{x}_0^\top]^\top$ and the target $\hat{\boldsymbol{x}}$. We will vary the number of layers N_L and the number of neurons per layer N_N, which we summarise in the model complexity (see Appendix A.1). Furthermore, we simulate datasets $\mathcal{D}$ of different sizes. Each data point consists of $\left(\left(t^{(i)}, \boldsymbol{x}_0^{(i)}, \boldsymbol{u}^{(i)}\right), \boldsymbol{x}^{(i)}\right)$. The optimiser will be the Limited Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) optimiser as explained in Section 2.3. Appendix A.1 provides further details on the setup. All in all, it is a basic regression problem solved with a vanilla NN.

The training process and result visualisation

We briefly explain how the results in the remainder of this chapter are generated and also explain their visualisation. This description is an illustration of the concepts introduced in Section 2.3. The training is performed by repeatedly applying the gradient of the loss function with respect to the parameters $\nabla_{\boldsymbol{\theta}}\mathcal{L}$. By tracking the loss on the training dataset $\mathcal{D}_{\text{Train}}$ over the epochs[8], we obtain a

[8]We use the term epoch for an iteration of the L-BFGS optimiser independent of the number of internal steps that the optimiser takes.

training loss $\mathcal{L}(\mathcal{D}_{\text{Train}})$ that monotonically decreases. If we at the same time evaluate the loss $\mathcal{L}$ on a separate dataset, the validation dataset $\mathcal{D}_{\text{Validation}}$, we obtain an estimate of the *generalisation error*. Ideally, we use the parameters $\boldsymbol{\theta}^*$ at the epoch E^* that marks the lowest validation loss $\mathcal{L}(\mathcal{D}_{\text{Validation}})$ as they provide empirically the best *generalisation*. We want to stop the training before the model starts to overfit the training data. To assess this model, we utilise another dataset, the test dataset $\mathcal{D}_{\text{Test}}$. Here, we compute the loss $\mathcal{L}$, but we can also assess how well the model performs on the performance metric $\mathcal{P}$. Why should we not use directly $\mathcal{D}_{\text{Validation}}$ for this assessment? By stopping the training at the point with the lowest validation loss $\mathcal{L}(\mathcal{D}_{\text{Validation}})$, the estimate becomes biased towards this particular validation dataset $\mathcal{D}_{\text{Validation}}$. With another validation dataset $\mathcal{D}_{\text{Validation}}$, we might have continued for a few more epochs. The use of the entirely independent test dataset avoids the induced bias and we can formulate more credible estimates for the loss and the performance metric – it is therefore imperative to not include the test dataset in any decision in the training process.

The next question is whether the performance was a lucky coincidence or is repeatable? To that end, we initialise the parameters $\boldsymbol{\theta}$ at epoch 0 to different values. This initialisation is a random sampling of the weight and bias values, the chosen distribution is highly important as discussed in [76]; the initialisation suggested by the authors has become a standard choice for feed-forward NNs, called Xavier Glorot's initialisation. By using different randomisation seeds in the sampling, we obtain different initial models. Entirely independent of each other, we train and select the models. Furthermore, the used training and validation datasets stem from different realisations of the data-generating process. In effect, we estimate the *expected generalisation error*. The testing is performed on the same (large) dataset for all models. This provides us with a collection of results that we can use to judge the repeatability of the obtained performance. Subsequently, we will often use boxplots for their representation. The box represents the range from the 25th to the 75th percentile and the black line inside the box indicates the median. The whiskers (the lines above and below the box) indicate the remaining range of observed values. If their length exceeds 1.5 times the interquartile range, observations are considered outliers and represented as points.

Analysing the Bias Variance Trade-off

We now repeat the above described process for different sizes of the training dataset and different model complexities, i.e., we test hypothesis classes with different representation capacities. The results will describe the BVTO (see section Section 2.3) for the above defined task $\mathcal{T}$. Such an analysis is computationally (extremely) expensive, which means that we can only conduct it in this detail for "small" problems. It allows us for a moment a view on the generalisation of NN models that is rarely attainable. For this reason, we need to keep in mind which quantities we usually have access to and which ones are usually unattainable. A helpful quantity for this analysis is the difference between the training loss and the testing loss, we will refer to it as the *generalisation gap*[9].

We begin by plotting the training loss $\mathcal{L}_x(\mathcal{D}_{\text{Train}})$ (grey) and the testing loss $\mathcal{L}_x(\mathcal{D}_{\text{Test}})$ (black) as a function of the model complexity in Figure 3.6. The left-most panel shows a model with a low model complexity, in the other panels we increase it. The x-axis shows the dataset size (on an approximately logarithmic scale). The (unsurprising) observation is that an increase in the dataset

[9]We note that this term can sometimes have slightly different definitions, e.g., comparing [77], [78]. The interested reader is referred to the more learning theoretical view on generalisation, a very active field of research, e.g., [64].

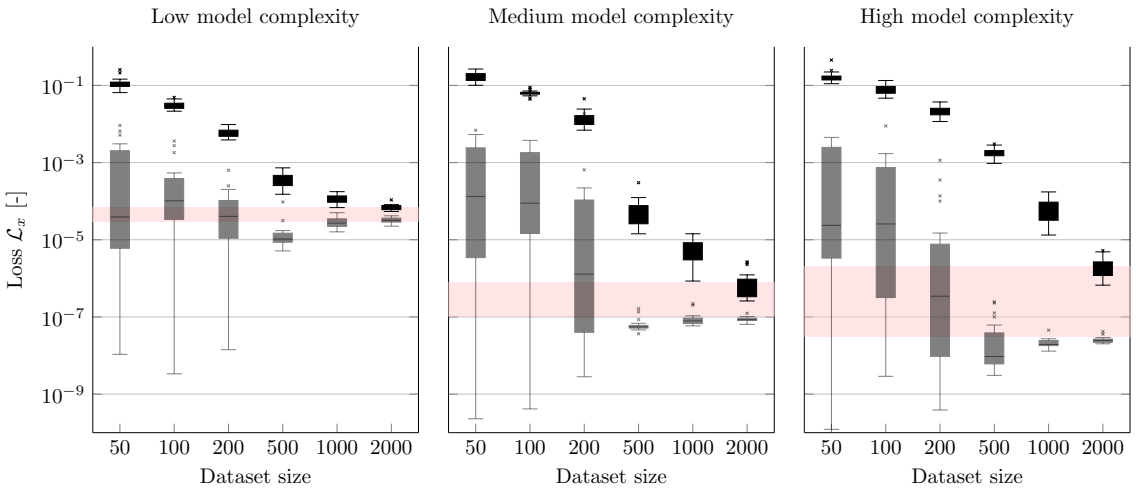


Figure 3.6: Training loss (grey) and testing loss (black) for different dataset size (x-axis) and model complexities (panels). The red shaded area indicates the generalisation gap for a large dataset.

size leads a decrease in the testing loss, i.e., the models generalise better. While the training loss has a large spread across different initialisations for small dataset sizes, it appears that the training and testing loss tend towards a limit value for larger datasets. We indicated the range of this limit value with the red shaded area. The associated model error can be attributed to the bias of the model when decomposing the error into bias and variance. If the bias is dominant, the model lacks representation capacity to fit the data better; more data points will not help. However, when the generalisation gap is not very small and run dependent, as it is the case for the smaller datasets, the error decomposition of the testing loss is not straightforward. Do we attribute the occurring error primarily to the bias or the variance in the model?

Based on the testing loss in the "large-dataset limit" we could perform a hypothetical decomposition of the error for a small dataset case into bias and variance. The result would be that the models are variance-dominated (the bias contribution is at most 10^{-4}) – the use of the logarithmic y-scale may distort the impression a bit. There are two inaccuracies in this reasoning: First, the bias is not always at its limit value, it can differ for low dataset cases. The second complicating fact is, that the testing error is also just an estimate, we need a sufficiently large test dataset for it to be accurate. A thorough statistical analysis would be needed for more precise statements. However, our intention here is to illustrate what it means if a model is variance-dominated or bias-dominated and that it is a trade-off, as the term BVTO implies.

We can in turn also fix the dataset size and observe what an increase in model complexity entails. The three panels in Figure 3.7 show the results. Starting from the smallest dataset, we observe that even though the training loss is decreasing significantly, the testing loss remains at relatively high levels, and even increases for hypothesis classes with higher model complexity and representation capacity. When increasing the dataset size as shown in the other two panels, the more complex models lead to much lower losses and the generalisation gap decreases as well. For the simpler models, the generalisation gap nearly vanishes. When talking about the BVTO, we implicitly want to answer the question of which model complexity to choose for a given dataset size to achieve good generalisation – without knowing the shown plots. In practice, we often simply choose the largest available dataset and then aim to find an hypothesis class with an appropriate model complexity.

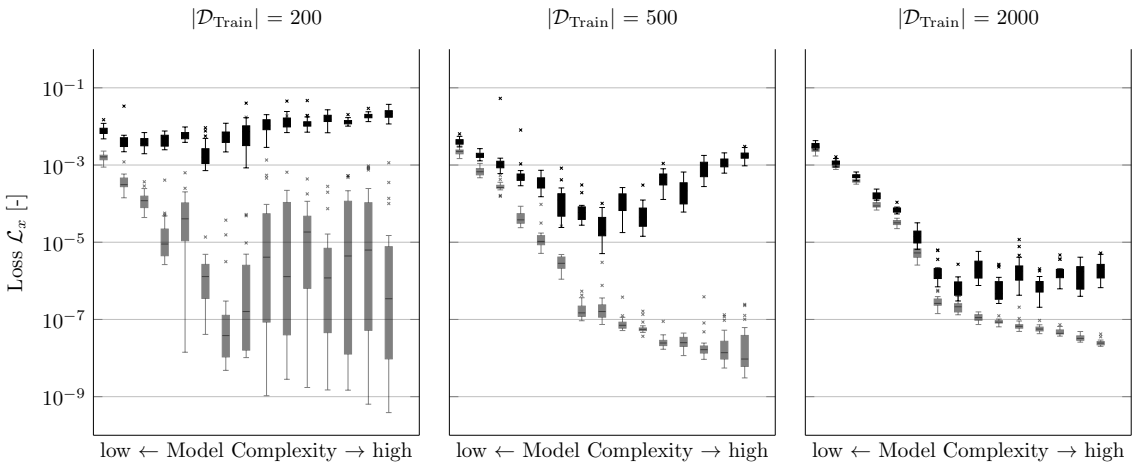


Figure 3.7: Training loss (grey) and testing loss (black) for different model complexities (x-axis) and dataset size (panels).

We can also depict the geometry of the BVTO in a three-dimensional fashion by plotting model complexity and dataset size on the x and y axes and the median loss from Figures 3.6 and 3.7 on the z axis. The colouring uses the logarithm of the loss values. In Figures 3.6 and 3.7, we

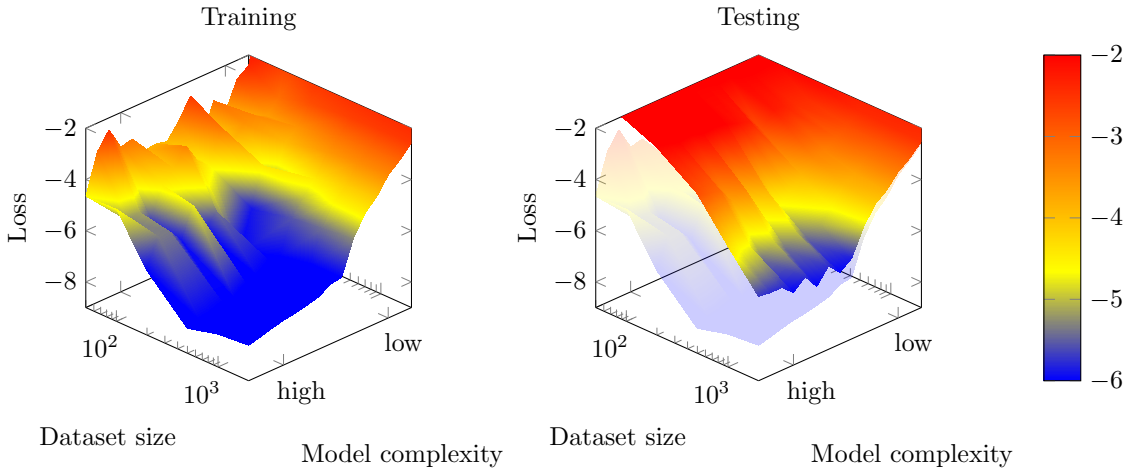


Figure 3.8: Bias-variance trade-off (BVTO) in the three dimensional space of the model complexity, the dataset size and the loss value (also indicated by the colouring). The opaque surface in the right plot corresponds to the training loss from the left plot.

effectively looked at cross-sections of Figure 3.8. The three-dimensional view makes it even clearer that well-fitted models (low testing loss) require both, sufficient representation capacity and large enough datasets. If we extended the plot to even larger models and datasets, we would continue to approach smaller and smaller test loss values.

A useful and simpler visualisation is to consider the testing loss in Figure 3.8 from a top view as shown in Figure 3.9. Here, we show the generalisation gap as level sets depicted by the black dashed lines. A value of -1 means, that the training loss is 10^{-1} times the testing loss. We can see, that the models in the upper left corner are bias dominated (low generalisation gap) and in the lower right corner variance dominated (large and inconsistent generalisation gap). At this point, it is important to point out that, ultimately, we care about a small enough testing loss. The

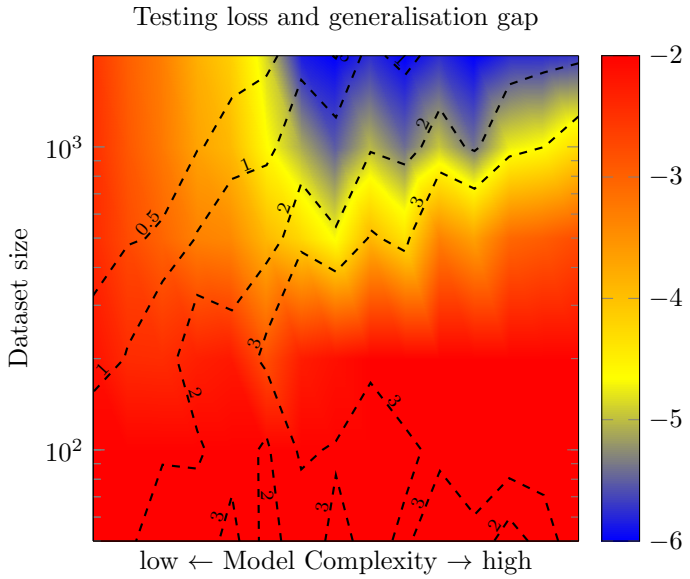


Figure 3.9: Top view onto Figure 3.8, i.e., the BVTO. The dashed lines indicate the generalisation gap of the bias variance trade-off. The colour scale shows the test loss.

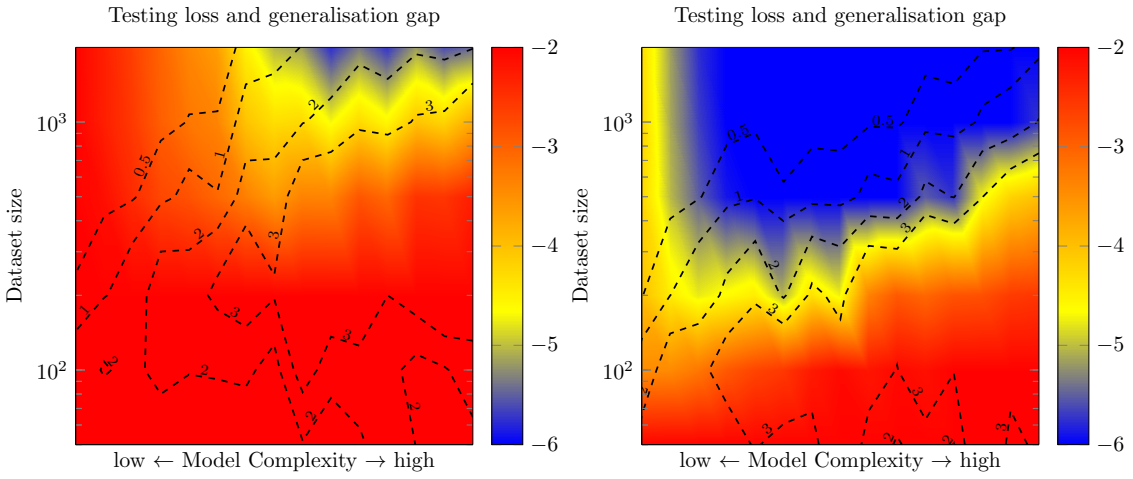
distinction between bias and variance related model error is a "construction". It helps, though, to understand if we should aim for reducing bias, i.e., increasing the model complexity, and, hence, the representation capacity, or for reducing variance, i.e., providing more data points. With a small generalisation gap, this attribution becomes clearer, i.e., the bias is dominating. For this reason we prefer a small generalisation gap. As we will see, *inductive biases* play an important role to achieve good generalisation and we will explore some in Section 3.5.

Exploring the BVTO – hyperparameter tuning and task-dependency

As mentioned earlier, to generate the shown BVTO plots, a very extensive analysis of the learning task is necessary and it is usually not feasible to obtain. If it was, we could simply select an appropriate model size and dataset size based on the testing loss requirements. In practice, hyperparameter tuning serves the role of obtaining a rough approximation of the BVTO. The hyperparameters can include the model complexity, e.g., depth and width of the NN, the dataset size, but they will also need to take hyperparameters from inductive biases into account. Examples are regularisation strengths and optimiser parameters such as the learning rate. Hyperparameter tuning requires the training of multiple models, therefore, the development of efficient search routines is an important topic for obtaining good models.

For some hyperparameters, in particular those of the optimisers, heuristics can inform suitable ranges. However, other settings will be dependent on the specific learning task $\mathcal{T}$. Figure 3.10 shows two tasks that are very related to the one in Figure 3.9, however, they result in very different BVTO plots. In Figure 3.10(a), we doubled the prediction horizon, i.e., we can predict larger time steps h. The region of low testing loss (blue) becomes much smaller which in turn will require more data points to obtain the same accuracy as for the original task. If we, in contrast, reduce the domain from which the initial conditions stem (Figure 3.10(b)), i.e., reduce $\mathcal{X}_0$ (we set $r < 0.5$, hence, the points are closer to the equilibrium state), the region of low testing loss increases dramatically. Furthermore, the generalisation gap becomes more favourable in this case. These

examples illustrate that the BVTO is task dependent and we can say that there exist different levels of *task difficulties*; a *simple* task can be solved accurately with few data points and a low representation capacity, for a *difficult* one, we require more powerful hypothesis classes and more data points. As an investigation of this "level of difficulty" would entail further increasing the already significant computational burden, we focus in the remained only on the initially specified task.



(a) Double prediction horizon, i.e., $t = 2\,\mathrm{s}$

(b) Half radius, i.e., $r(\boldsymbol{x}_0) \leq 0.5$

Figure 3.10: Impact of altering the learning task $\mathcal{T}$ on the BVTO.

3.5 Including domain knowledge - PINNs

In this section, we present different options to incorporate domain knowledge from the dynamical system; we aim to include *inductive biases* that can be informed by the physics. Ideally, these inductive biases alter the BVTO in our favour, such that we reliably obtain low testing losses and generalisation gaps. Based on the central components of a learning algorithm, laid out in Chapter 2, there are four potential "points of entry": The loss function $\mathcal{L}$, the dataset $\mathcal{D}$, the hypothesis class h_θ, and the optimiser. The following paragraphs provide the conceptual ideas and mathematical formulations. In Section 3.6, we show their effects on some example cases.

3.5.1 Physics-based loss functions

This section explores how to incorporate the physical knowledge into the learning, specifically into the loss function $\mathcal{L}$, i.e., altering the training objective. This approach is based on the work of Raissi, Perdikaris and Karniadakis [26] who revived and extended earlier work [67] under the term Physics-Informed Neural Networks (PINNs). Section 3.7 provides further context on PINNs, we now focus on the formulation.

In the following, we change neither the performance metric $\mathcal{P}$ nor the use of $\mathcal{L}_x$ (3.7c) as a metric to compare models. Instead, we add additional terms to the loss $\mathcal{L}$, i.e., the objective function during training. The intuition behind these added terms is that the NN should not only fit the data points well, but also the tangent with respect to time, i.e., the temporal derivative. We essentially

incorporate the defining property of a flow into the training procedure, visually speaking we utilise the vectorfield.

We start with a dataset $\mathcal{D}$ that stems from simulation of trajectories, and we form the loss according to (3.7c)

$$\mathcal{L}_x(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)} \right\|_{T,2}^2 . \tag{3.11}$$

dt-loss

In a first step, we propose to incorporate the underlying physics in a "dt-loss" as follows: We evaluate the left-hand side (lhs) and right-hand side (rhs) of the differential equation

$$\underbrace{\frac{d}{dt}\boldsymbol{x}}_{\text{lhs}} = \underbrace{\boldsymbol{f}\left(t, \boldsymbol{x}, \boldsymbol{u}\right)}_{\text{rhs}}$$

based on the simulated data points $\boldsymbol{x}^{(i)}$ and the approximated data points $\hat{\boldsymbol{x}}^{(i)}$. The norm of their mismatch is then computed in the dimensionless states[10], indicated by the subscript T, and forms two additional loss terms $\mathcal{L}_{lhs}$ and $\mathcal{L}_{rhs}$

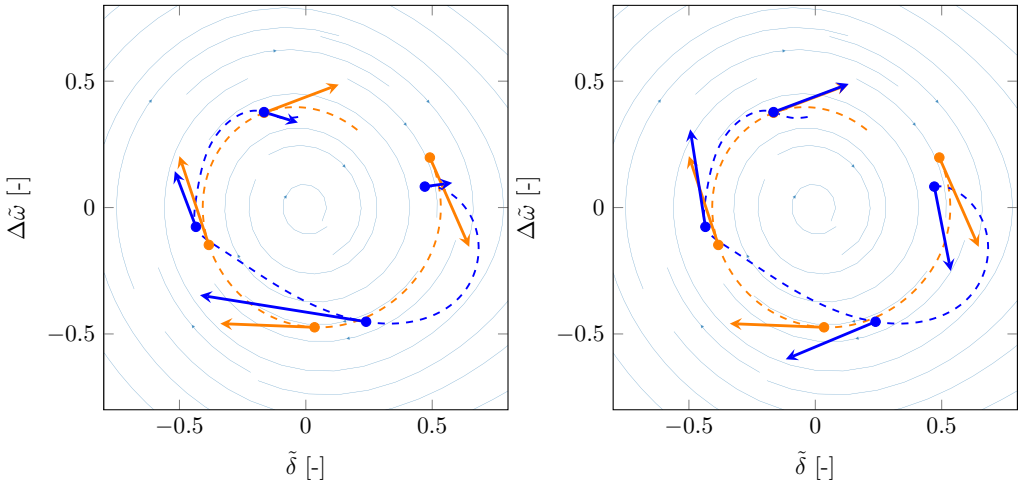
$$\mathcal{L}_{lhs}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \frac{d}{dt}\boldsymbol{x}^{(i)} - \frac{d}{dt}\hat{\boldsymbol{x}}^{(i)} \right\|_{T,2}^2 \tag{3.12}$$

$$\mathcal{L}_{rhs}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \boldsymbol{f}\left(t^{(i)}, \boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}\right) - \boldsymbol{f}\left(t^{(i)}, \hat{\boldsymbol{x}}^{(i)}, \boldsymbol{u}^{(i)}\right) \right\|_{T,2}^2 . \tag{3.13}$$

Figure 3.11 illustrates the interpretation in the vectorfield, as the losses can be seen as differences of vectors. The orange curve corresponds to a true trajectory, and we evaluate it at four points $\boldsymbol{x}$ (coloured dots). The NN-based prediction of the trajectory is shown in blue, also evaluated at four points $\hat{\boldsymbol{x}}$. $\mathcal{L}_x$ (3.11) relates to the distance between the states. Here, the use of the dimensionless states $\tilde{\boldsymbol{x}}$ shows its importance as the *distance* is now clearly defined. By computing the vector that is tangent to the trajectory, we obtain the vectors at the points shown in Figure 3.11(a). Their length indicates how fast the state moves along this trajectory. The difference between the vectors corresponds to $\mathcal{L}_{lhs}$. For $\mathcal{L}_{rhs}$ we also calculate the difference between two vectors, but these stem from the vectorfield, i.e., the update function $\boldsymbol{f}(\cdot)$ evaluated at the true or predicted point, see Figure 3.11(b). For the true trajectory the vectors in Figures 3.11(a) and 3.11(b) are identical by definition[11], but this is not true for the NN approximation. The following thought shall further clarify the difference between the two losses: For $\mathcal{L}_{lhs}$ the trajectories could be offset by any value, the loss would remain unchanged as it penalises the mismatch of the derivatives of the trajectories. In contrast, $\mathcal{L}_{rhs}$ penalises a mismatch in the location in the vectorfield. This entails that the two are not identical and can vary independently, even to the extent that one equals 0 while the other does not. It also means, that $\mathcal{L}_{rhs}$ acts more directly on the optimisation and has overlapping information content with $\mathcal{L}_x$, whereas $\mathcal{L}_{lhs}$ adds a significant regularisation because it incorporates information of the surrounding of the data point as its derivative describes the local neighbourhood.

[10] The transformation for $\frac{d}{dt}\boldsymbol{x}$ can be derived by computing the temporal derivative of $\boldsymbol{T}$ (3.4).

[11] In the implementation, we therefore replace $\frac{d}{dt}\boldsymbol{x}^{(i)}$ in $\mathcal{L}_{lhs}$ by $\boldsymbol{f}\left(t^{(i)}, \boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}\right)$.



(a) $\mathcal{L}_{lhs}$ **(b)** $\mathcal{L}_{rhs}$

Figure 3.11: Illustration of the interpretation of the loss terms $\mathcal{L}_{lhs}$ and $\mathcal{L}_{rhs}$ in the state space. The orange line corresponds to the true trajectory and the blue line to a NN-based prediction.

Physics-loss

These regularisation terms $\mathcal{L}_{lhs}$ and $\mathcal{L}_{rhs}$ rely on a comparison based on the training dataset. The fundamental idea of PINNs as introduced in [26] is that we can add another term, $\mathcal{L}_c$, which compares the left-hand side and right-hand side of the update function, but now both evaluated on the prediction $\hat{\boldsymbol{x}}^{(i)}$

$$\mathcal{L}_c(\mathcal{D}_c) = \frac{1}{|\mathcal{D}_c|} \sum_{i=1}^{|\mathcal{D}_c|} \left\| \frac{d}{dt}\hat{\boldsymbol{x}}^{(i)} - \boldsymbol{f}\left(t^{(i)}, \hat{\boldsymbol{x}}^{(i)}, \boldsymbol{u}^{(i)}\right) \right\|_{T,2}^2 . \tag{3.14}$$

We adapt the original formulation to the ODE formulation and the transformed state space, the conceptual idea is identical. Figure 3.12 illustrates this loss term again on the vectorfield. In contrast to the previously added terms $\mathcal{L}_{lhs}$ and $\mathcal{L}_{rhs}$, we are now independent of the simulated trajectory and data points; we can evaluate $\mathcal{L}_c$ on any point in the input domain (it is even possible outside the original domain bounds). We dedicate a special dataset to it, $\mathcal{D}_c$, which contains all of these *collocation points*[12]

$$\mathcal{D}_c := \left\{ \left(\left(t^{(1)}, \boldsymbol{x}_0^{(1)}, \boldsymbol{u}^{(1)} \right), \mathbf{0} \right), \ldots, \left(\left(t^{(N)}, \boldsymbol{x}_0^{(N)}, \boldsymbol{u}^{(N)} \right), \mathbf{0} \right) \right\}. \tag{3.15}$$

It is crucial to stress that $\mathcal{L}_c = 0$ does not imply that the predicted flow $\hat{\Phi}$ is correct, it simply tells that the predictions are in line with the vectorfield. Formally, this can also be seen directly from (3.15), as the data-generating function necessary to create such a dataset is not bijective, i.e., not invertible. The target $\mathbf{0}$ can be reached from infinitely many input points. Because we know the target for any input, we can generate datasets $\mathcal{D}_c$ of arbitrary size without running a single simulation.

[12]The name indicates the connection to numerical collocation methods which are constructed around the evaluation of the consistency of the prediction rather than by comparing the prediction against a target, see, e.g., [12].

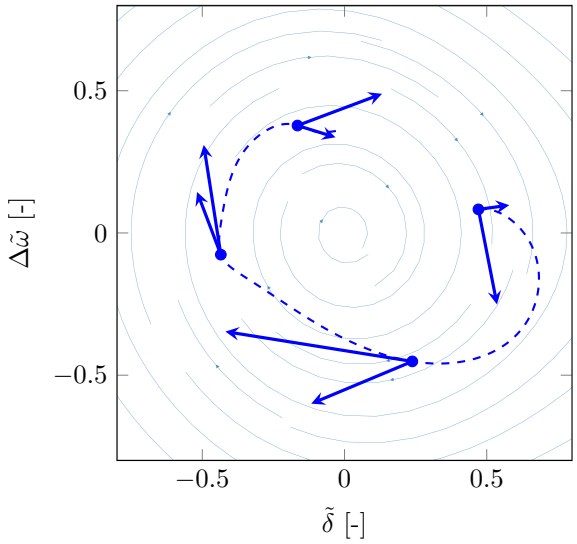


Figure 3.12: Illustration of the interpretation of the physics loss $\mathcal{L}_c$ in state space. The blue line to a NN-based prediction and the two arrows represent $\frac{d}{dt}\hat{\boldsymbol{x}}$ and $\boldsymbol{f}(\hat{\boldsymbol{x}})$.

Total training loss

We add all loss terms together into the total training loss $\mathcal{L}$

$$\mathcal{L} = \mathcal{L}_x + \alpha_{dt}(\mathcal{L}_{lhs} + \mathcal{L}_{rhs}) + \alpha_c \mathcal{L}_c. \tag{3.16}$$

and we once more encounter a multi-objective optimisation problem. We reduce it to a single objective by weighting the different physics-based loss terms, i.e., the dt-loss ($\mathcal{L}_{lhs} + \mathcal{L}_{rhs}$) and the physics-loss $\mathcal{L}_c$, with factors α_{dt} and α_c which become hyperparameter of the problem. The fact that $\mathcal{L}_c = 0$ does not lead to a unique solution, whereas $\mathcal{L}_x$ as well as $\mathcal{L}_{lhs}$ in combination with $\mathcal{L}_{rhs}$ do, can produce undesired training outcomes if α_c is too large. At the same time, when it is too small, we loose the effect of the physics-based regularisation. To balance these considerations, we utilise an adaptive scheme based on the current epoch E. Starting from an initial weighting factor $\alpha_{c,0}$, we increase the weighting according to

$$\alpha_c(E) = \min\left(\alpha_{c,\max}; \alpha_{c,0}\ 10^{E/E'}\right) \tag{3.17}$$

up to $\alpha_{c,\max}$. The hyperparameter E' determines the speed of this fade-in. In [79], the authors propose a learning rate annealing that aims at balancing the gradient of the different loss terms with respect to the NN parameters. In Section 4.4, we provide a probabilistic view on this weighting problem.

3.5.2 Dataset design

The *data-generating process* for task $\mathcal{T}$ becomes the flow Φ itself as there is no noise present, i.e., $\varepsilon = 0$. The relation between the input $(t, \boldsymbol{x}_0, \boldsymbol{u})$ and the output $(\boldsymbol{x})$ is per definition described by the flow Φ

$$\boldsymbol{x} = \Phi(\boldsymbol{x}_0, t; \boldsymbol{u}). \tag{3.18}$$

For this process, we formulate the *data-generating distribution* $p_{data}(t, \boldsymbol{x}_0, \boldsymbol{u})$, from which the dataset will stem

$$p_{data} = p(t)\, p(\boldsymbol{x}_0)\, p(\boldsymbol{u}). \tag{3.19}$$

This may seem either trivial or just needlessly formal. The reason to start from this point is that it motivates how the design of the dataset can be used to influence the learning outcomes. With the term *design* we refer to the opportunity to choose the probability distributions of the input variables, which in turn shapes p_{data}. We sample a dataset $\mathcal{D}$ from p_{data}, which forms the experience $\mathcal{E}$ based on which the algorithm learns. Intuitively, it matters what experience the model is exposed to and what we aim for. As we can decide which experience we provide the learning algorithm with, the question naturally arises: How should we choose the distributions of the input variables to obtain a good performance $\mathcal{P}$, i.e., a small maximum absolute error? The restriction of the input domain for the task $\mathcal{T}$ provides bounds for the probability distributions, i.e., their support, but not the probability density functions. In the following, we will suggest a few choices for $p(t)$ and $p(\boldsymbol{x}_0)$ based on considerations about $\mathcal{T}$ and $\mathcal{P}$. We do not cover $p(\boldsymbol{u})$ explicitly, as it is not varied in Section 3.6, but similar arguments as for $p(t)$ and $p(\boldsymbol{x}_0)$ could be made.

Sampling time $p(t)$

In the task $\mathcal{T}$, we defined the input domain of interest for the time, or more precisely, the time step h as $[0; h_{\max}]$. The choice of this domain, i.e., values between 0 and $h_{\max}$ is specific to the task $\mathcal{T}$ and therefore fixed. Needless to say though, that this choice has major implications on the learning as we saw for the BVTO in Section 3.4 and as we will discuss in Section 3.7. Given $h_{\max}$, a straightforward choice for $p(t)$ is to use a uniform distribution $\mathcal{U}(\min, \max)$ between the minimum and maximum value

$$t \sim \mathcal{U}(0, h_{\max}). \tag{3.20}$$

From the comparison with conventional numerical schemes, a desirable property for the NN-based flow approximators might be a time step dependent error. From that perspective, larger time steps could be allowed to have larger errors than very small time steps, it is the concept of numerical consistency. If we increase the density of points in the region where we aim for small errors, we can induce such a characteristic. To this end, we test a log-uniform distribution in t, i.e., the logarithm of t is distributed uniformly between a lower and an upper bound.

$$\log t \sim \mathcal{U}(0.001, h_{\max}). \tag{3.21}$$

For the lower bound we need to choose a positive value, e.g., 0.001 s.

Sampling the initial condition $p(\boldsymbol{x}_0)$

The sampling of the initial condition is more intricate as we need to consider the different elements $x_{0,i}$ of the state $\boldsymbol{x}$. If we treat them separately, we can simply choose a uniform distribution $\mathcal{U}$ with bounds $\underline{\mathrm{x}}_{0,i}, \bar{x}_{0,i}$ for each element of the state and multiply their probabilities

$$x_{0,i} \sim \mathcal{U}(\underline{\mathrm{x}}_{0,i}, \bar{x}_{0,i}) \tag{3.22}$$

$$p(\boldsymbol{x}_0) = \prod_{i=1}^{n} p(x_{0,i}). \tag{3.23}$$

In effect, we sample from an n-dimensional hypercube, where n is the number of states.

At this point, the considerations that led to the formulation of the dimensionless state space (see Section 3.3) occur again: First, the concept of a state space highlights that the *state* $\boldsymbol{x}$ of a system should be considered with all elements of the state at once. Second, the shape that defines the

critical energy looks more like a circle than a square. Based on this geometric observation, we sample the entire state $\boldsymbol{x}$ simultaneous and more alike sampling in a unit-ball defined by the L^2-norm[13]. For the sampling we split the sampling process into sampling points $\boldsymbol{x}_0'$ on an n-dimensional sphere ($\|\boldsymbol{x}_0'\|_{T,2} = 1$) using the multi-variate normal distribution $\mathcal{N}$ with I being the identity matrix

$$\boldsymbol{x}_0' \sim \mathcal{N}(\boldsymbol{0}, I) \tag{3.24}$$

$$p\left(\boldsymbol{T}\left(\boldsymbol{x}_0\right)\right) = p(r)\frac{\boldsymbol{x}_0'}{\|\boldsymbol{x}_0'\|_2} \tag{3.25}$$

and then scale these points by r, i.e., radius of the sphere. As we describe the probability distribution in the dimensionless state space, the radius r is equivalent to the definition in (3.8). For the present case with $n = 2$, this sphere is a unit circle. If we sample the radius according to a uniform distribution $\mathcal{U}(0, r_{\max})$, we will obtain more points towards the centre of the circle, i.e., close to the stable equilibrium point $\boldsymbol{x}_{eq}$. If we sample such that r^2 is uniformly distributed, the area will be equally covered with data points. This leads to two choices for the probability distribution $p(r)$, which we will refer to as *radial* and *annular* sampling of the initial condition:

$$\text{Radial:} \qquad r \sim \mathcal{U}(0, r_{\max}) \tag{3.26}$$

$$\text{Annular:} \qquad r^2 \sim \mathcal{U}(0, r_{\max}) \tag{3.27}$$

We will compare these strategies to the sampling defined by (3.23), we call it *square* sampling. For a "fair" comparison the resulting input domain $\mathcal{X}_0$ should cover the same area (or volume for $n > 2$). Therefore, we set the limits to $\boldsymbol{T}\left(\underline{x}_{0,i}\right), \boldsymbol{T}\left(\bar{x}_{0,i}\right) = \pm\sqrt{\pi/4}r_{\max}$.

Sampling choices for testing model performance

We not only need to sample a training dataset $\mathcal{D}_{\text{Train}}$ for the learning algorithm, but also a testing dataset $\mathcal{D}_{\text{Test}}$ on which we evaluate the performance $\mathcal{P}$. For the maximum absolute error, for instance, the performance formulates as

$$\mathcal{P}^{\infty} = \left\|\hat{\Phi}(t, \boldsymbol{x}_0, \boldsymbol{u}) - \Phi(t, \boldsymbol{x}_0, \boldsymbol{u})\right\|_{\infty}, \qquad t \in [0, t_{\max}],\ \boldsymbol{x}_0 \in \mathcal{X}_0 \tag{3.28}$$

over the entire combined input domain of t, $\boldsymbol{x}_0$, and $\boldsymbol{u}$. The empirical performance $\mathcal{P}$ that we obtain from a test dataset, will always underestimate the true value $\mathcal{P}^{\infty}$ unless we exactly found the worst-performing point. If we have a very large number of points in the testing dataset, the estimation error $|\mathcal{P} - \mathcal{P}^{\infty}|$ will tend towards 0, however, for smaller datasets a significant estimation error can occur. Its size will depend on the sampling strategy, i.e., p_{data}, taken for the test dataset. Intuitively speaking, it depends on how well p_{data} covers the region where the flow approximation $\hat{\Phi}$ performs worst. Where this region is, however, depends itself on p_{data} of the training dataset and the training algorithm overall.

Weighting data points

An alternative way of *designing* the dataset can be achieved by weighting the importance of points. Each point in the dataset is assigned a weight and the loss functions ($\mathcal{L}_x$, $\mathcal{L}_{rhs}$, $\mathcal{L}_{lhs}$, $\mathcal{L}_c$) then calculate the weighted average over the loss per point instead of their mean. An advantage of such an approach is that the weights can be adjusted during the training, e.g., based on the collocation

[13]The hypercube from above is also a unit-ball when using the L^{∞}-norm, hence, (3.23) could also be derived based on the presented argument.

loss as in [80]. However, the choice of the weights can also lead to unintended biases in the resulting models. From a probabilistic view, the weights basically represent an importance distribution that we convolute with the sampling distribution to alter p_{data}. If we have a well-informed idea, e.g., from physics, what the importance distribution shall look like, it can help to provide the learning algorithm with experience $\mathcal{E}$ that leads to a good performance $\mathcal{P}$. We have not implemented nor tested this approach to avoid another layer of complexity as the weighting will interact with the choice of p_{data}.

3.5.3 Architecture of the hypothesis class

A central element of a well-specified learning process is the form of the hypothesis class h_θ. It allows us to include properties into the resulting approximation that shall be fulfilled by construction. One can think of it as hard constraints that cannot be violated or of symmetries and invariances that need to hold. The authors in [81] analyse these aspects from a geometric point of view and term it *Geometric Deep Learning*. We do not go into any details of this novel viewpoint, but it certainly inspired seeking the close connecting to the *geometry* of the state space and the vectorfield. Furthermore, the presence of geometric numerical integration (see e.g. [12]) hints at the importance of geometry when approximating the flow of a dynamical system.

For the given problem, we restrict the selection to models based on the feed-forward NN in (2.15). We add the transformation $\boldsymbol{T}$ for the input of the initial condition and its inverse $\boldsymbol{T}^{-1}$ for the output layer:

$$\boldsymbol{z}_0 = [h, \boldsymbol{T}(\boldsymbol{x}_0)^\top, \boldsymbol{u}^\top] \tag{3.29a}$$

$$\boldsymbol{z}_{k+1} = \sigma\left(\boldsymbol{W}^{(k)}\boldsymbol{z}^{(k)} + \boldsymbol{b}^{(k)}\right) \qquad k = 0, 1, ..., K-1 \tag{3.29b}$$

$$\hat{\boldsymbol{x}} = \boldsymbol{T}^{-1}\left(\boldsymbol{W}^{(K)}\boldsymbol{z}^{(K)} + \boldsymbol{b}^{(K)}\right) \tag{3.29c}$$

This forms the basis for the following architectures[14].

A first step, that [67] suggests, is to alter the final layer to include the initial condition, closely resembling the integral formulation of an ODE or the learning of a residual

$$\text{Residual:} \qquad \hat{\boldsymbol{x}} = \boldsymbol{x}_0 + \boldsymbol{T}^{-1}\left(\boldsymbol{W}^{(K)}\boldsymbol{z}^{(K)} + \boldsymbol{b}^{(K)}\right). \tag{3.30}$$

One can even enforce this constraint by including the time step size h explicitly, which will ensure consistency of the approximator as discussed in 3.2.1

$$\text{Residual A:} \qquad \hat{\boldsymbol{x}} = \boldsymbol{x}_0 + h\boldsymbol{T}^{-1}\left(\boldsymbol{W}^{(K)}\boldsymbol{z}^{(K)} + \boldsymbol{b}^{(K)}\right). \tag{3.31}$$

In the same manner, we can construct a function that obeys also $\boldsymbol{f}(0, \boldsymbol{x}_0, \boldsymbol{u})$, i.e., the tangent at the initial condition

$$\text{Residual B:} \qquad \hat{\boldsymbol{x}} = \boldsymbol{x}_0 + h\left(\boldsymbol{f}(0, \boldsymbol{x}_0, \boldsymbol{u}) + h\boldsymbol{T}^{-1}\left(\boldsymbol{W}^{(K)}\boldsymbol{z}^{(K)} + \boldsymbol{b}^{(K)}\right)\right). \tag{3.32}$$

All these changes, suggested in [67], do not require any adaptation of the loss formulation as the output is always $\hat{\boldsymbol{x}}$. However, the target probability distribution, that the NN learns, does change and hence also the optimisation landscape and the learned parameters $\boldsymbol{\theta}$.

[14] In the implementation, we also perform a standardisation of the input vector z_0. This usually helps the training process as otherwise vanishing gradients can easily occur due to saturation of the activation function.

IRK-PINNs

We can seek an even closer link to the numerical RK schemes by predicting $\hat{\boldsymbol{x}}$ with the intermediate step of computing the ν stages $\boldsymbol{k}^j$ of an implicit Runge-Kutta (IRK) scheme, proposed in [26]. For an explicit scheme this would provide only (if at all) a marginal benefit compared to directly applying the scheme. Computing the stages of an IRK scheme, however, requires the solution of an implicit non-linear system of equations. Therefore, an approximation of the solution to this implicit problem can result in a significant speed-up while profiting from the numerically desirable properties. The only change we have to make is, that the last layer predicts the stages of the IRK scheme $\hat{\boldsymbol{k}}^j$

$$\hat{\boldsymbol{x}} = \boldsymbol{x}_0 + h \sum_{j=1}^{\nu} b_j \boldsymbol{f}\left(t_0 + c_i h, \hat{\boldsymbol{k}}^j\right) \tag{3.33a}$$

$$\hat{\boldsymbol{k}}^j = \boldsymbol{T}^{-1}\left(\boldsymbol{W}^{(K)}\boldsymbol{z}^{(K)} + \boldsymbol{b}^{(K)}\right) \tag{3.33b}$$

To train such an IRK-PINN we can rely on the usual loss term $\mathcal{L}_x$. However, we can also add a collocation-based loss function that penalises mismatches in the definitions of the RK stages, i.e., of (2.10), when evaluated with the approximations $\hat{\boldsymbol{k}}^j$:

$$\mathcal{L}_{c,IRK} = \frac{1}{|\mathcal{D}_c|}\sum_{i=1}^{|\mathcal{D}_c|}\left\|\hat{\boldsymbol{k}}^{j^{(i)}} - \boldsymbol{x}_0^{(i)} - h^{(i)}\sum_{m=1}^{\nu} a_{j,m}\boldsymbol{f}\left(t_0 + c_m h^{(i)}, \hat{\boldsymbol{k}}^{m^{(i)}}\right)\right\|_{T,2}^2 . \tag{3.34}$$

This loss only becomes 0, when we have found a solution to the implicit problem defined by an IRK-scheme. The initial formulation by [26] used a fixed time step size h. We propose to include the time step size h as an input. Thereby, we obtain a full flow approximator scheme $\hat{\Phi}$ instead of only approximating specific flow maps $\hat{\phi}_t$. Furthermore, it enables the use of the other physics-based loss functions $\mathcal{L}_{lhs}$, $\mathcal{L}_{rhs}$, and $\mathcal{L}_c$. They can be evaluated independently of $\mathcal{L}_{c,IRK}$.

3.5.4 Optimiser

The last "ingredient" of the ML setup is the optimiser. Optimisers are known to have some influence on the training outcomes, i.e., they introduce some inductive bias [65]. However, only a handful of optimisers are popular as they are usually rather problem agnostic and applicable to a variety of problems. The choice is often determined by the available computational resources. Two popular and possible choices are the Adam and L-BFGS optimisers. To limit scope, we can provide merely experimental insights in how hyperparameters affect the process and performance. A theoretical conceptualisation is beyond the scope of this thesis. Potential insights regarding the optimisation landscape could be valuable for improving the choices in the previous three subsections - the dimensionless coordinates exemplify this. Achieving performance improvements by directly integrating domain knowledge into the optimisation algorithm is more difficult, given the highly optimised implementations. Specific adaptations of the optimisation algorithms for PINNs might be possible though.

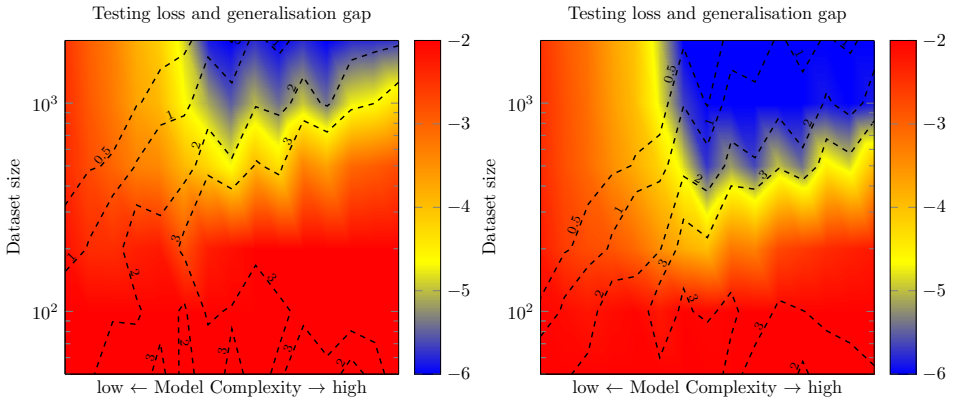
3.6 Exploring the space of physics-informed approaches

The above listed choices for informing the learning process could be combined in many ways, and interactions between them are to be expected. A full analysis of all possible combinations would require the exploration of too many combinations. Therefore, we present the effects of the inductive

biases and their interactions on a few selected examples. The physics-agnostic approach serves mostly as the common reference case, and we want to show how the *inductive biases*, that we introduced in the previous section, influence the observed Bias Variance Trade-off (BVTO). By building a good understanding of "local" effects, we can start to hypothesise on potentially useful combinations without having a rigid theory yet. Thereby, we aim to avoid running extensively many model trainings – in total we already trained more than 10 000 models for this section. Reducing this number is critical when moving to more intricate setups and dynamical systems, as the computational burden will be too high for an investigation of this size.

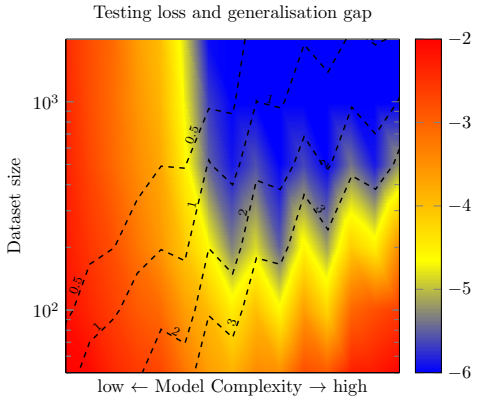
3.6.1 Regular points vs collocation points

We begin by analysing the effects of including the governing ODEs in the loss function. This can either be based on including the dt-loss, i.e., $\mathcal{L}_{lhs} + \mathcal{L}_{rhs}$ evaluated on the simulated dataset, or based on the collocation-based term $\mathcal{L}_c$, see Section 3.5.1 for their definition. We will use the terms *dtNN* and *PINN* to refer to them. A NN without the additional loss terms will be called *vanilla NN*. The overall setup is identical to the analysis of the BVTO in Section 3.4. The hyperparameter settings can be found in the appendix.



(a) vanilla NN

(b) dtNN



(c) PINN

Figure 3.13: Illustration of the BVTO for vanilla NNs, dtNNs, and PINNs. The colouring represents the loss on the test dataset ($\log_{10} \mathcal{L}_x(\mathcal{D}_{\text{Test}})$) and the dashed lines indicate the generalisation gap ($\log_{10}(\mathcal{L}_x(\mathcal{D}_{\text{Train}})/\mathcal{L}_x(\mathcal{D}_{\text{Test}}))$)

Figure 3.13 shows the resulting BVTO plots, the one labelled vanilla NN is identical to Figure 3.9. We clearly see that dtNNs and PINNs extend the region of low loss (blue) significantly, while also extending the region where the generalisation gap is low, i.e., the region in the left upper corner above the black dashed line. Both these trends are desirable as they yield better performing models (measured by the loss) and more confidence in the generalisation (measured by the generalisation gap). The following will go into more depth of how the additional loss terms affect the BVTO.

To start, we consider different cross-sections of the BVTO in Figure 3.14, which has the same interpretation as Figure 3.6; the three columns show the training loss in the upper panels and the testing loss in the lower panels for three model complexities. The x-axis shows the increasing dataset sizes. The boxplots represent the distribution over 20 training runs and their colour indicates the three NN types, vanilla NNs, dtNNs, and PINNs. In the lower panels, we additionally plot (very lightly coloured) the training loss from the upper panels. This will help to illustrate the generalisation gap that we will analyse in the following. We begin with the general observation,

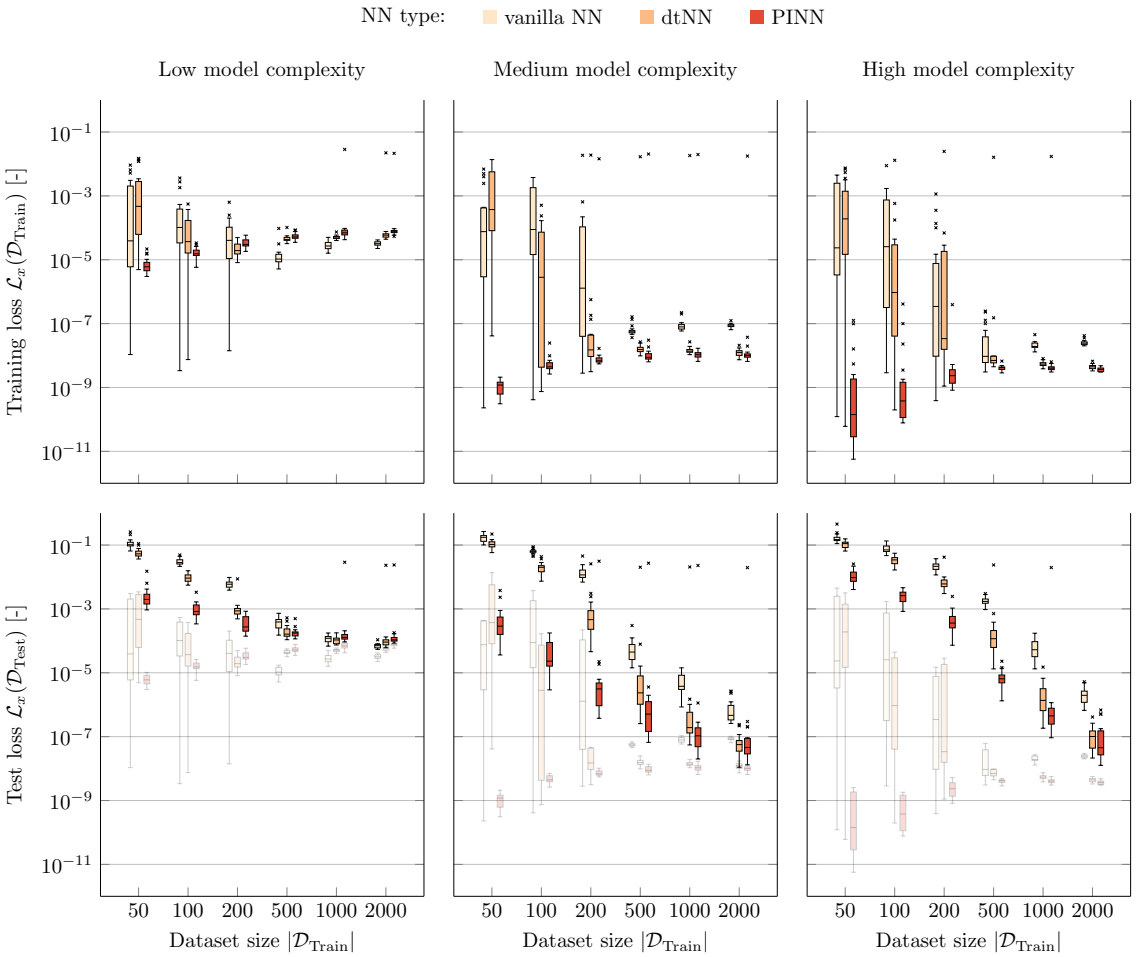


Figure 3.14: Generalisation gap between the training and testing loss for different model complexities (columns) and dataset sizes (x-axis). The colour of the boxplots corresponds to the NN type (vanilla NN, dtNN, PINN).

that PINNs deliver the lowest test loss across nearly all dataset sizes and model complexities when compared to the corresponding dtNN or vanilla NN. In the comparison of dtNNs and vanilla NNs, the former leads to lower test losses. For all three model complexities, i.e., the columns, the training

loss of the vanilla NNs and dtNNs shows a wide spread when the training dataset size is small; it is the same observation as in Figure 3.6. So while the dt-loss term does not alleviate this problem of high variance in the models, the training loss of PINNs is even for small dataset size much lower and more consistent. The values also differ much less with respect to the test loss for a large dataset size – here the model becomes bias dominated – in most cases less than by a factor of 100. This means that the training loss provides a much better estimate of the size of the model bias. In contrast, the training losses of the vanilla NNs and dtNNs only become "informative" for dataset sizes of 500 and above. This can also be seen in the "weird" patterns of the generalisation gap at the bottom of the BVTO plots in Figure 3.13. While the additional loss terms of dtNNs and PINNs are particularly helpful for a low and medium model complexity, the generalisation gap for a high model complexity remains significant except for the largest dataset size. Even in terms of the absolute levels of the test loss, the medium complexity models perform better. The reason is that the additional regularisation does not sufficiently restrict the representation capacity which results in a variance-dominated model. For the low model complexity, we encounter the opposite, a bias-dominated model.

As we can see for the model of medium complexity, PINNs can significantly improve the accuracy for small dataset sizes. As we add the collocation points (here 500) on top of the simulated data points, we obtain a better coverage of the domain to control the learning. Could we add more collocation points to improve performance further? In Figure 3.15 we use a dataset $\mathcal{D}_{\text{Train}}$ with 200 points and then alter the number of additional collocation points represented by the different coloured boxplots. Along the x-axis, we increase the model complexity. For the high complexity models, we do see a substantial improvement of the test loss by adding collocation points. Furthermore, the generalisation gap is also reduced. However, for high complexity models it appears to be difficult to achieve a generalisation gap as low as for low model complexities. This difficulty or lack of efficiency relates back to the nature of collocation points. They penalise solutions that are not aligned with physics, i.e., those solutions that contradict the vectorfield. In contrast, predictions that do not match the desired target but are physically consistent, do not lead to a loss. A simple example is when a PINN predicts that the trajectory equals the equilibrium state – this will be perfectly in line with the ODEs but it may contradict the supplied data points.

The above presented results illustrate the benefit of adding the dt-loss ($\mathcal{L}_{rhs} + \mathcal{L}_{lhs}$) based on the simulated dataset as well as $\mathcal{L}_c$ based on collocation points. The results suggest that PINNs provide a powerful inductive bias, in particular for models of medium complexity. These models have a low enough bias to be attractive in terms of the achievable test loss, whilst requiring only small simulated datasets along the collocation dataset to yield good training results.

3.6.2 Learning only from collocation points - PINNs and IRK-PINNs

In the limit, we can learn the flow Φ entirely from collocation points[15]; the total loss becomes equivalent to the physics-loss $\mathcal{L}_c$, we omit the regular data loss $\mathcal{L}_x$ and the dt-loss ($\mathcal{L}_{lhs}$+$\mathcal{L}_{rhs}$). We need enough collocation points, though, to ensure that the vectorfield is closely matched everywhere on the training domain. The difficulty lies in the fact that the solution is dependent on the solution

[15]When solving PDEs with PINNs as it was introduced in [26], this is the normal case for forward problems as we cannot simply compute "parts" of the solution and provide them as data points. For approximating a flow, in contrast, the possibility to simulate single trajectories allows us to easily include data points. Learning purely based on collocation points becomes, therefore, a more hypothetical exercise, as the use of simulated data points greatly accelerates the training. We come back to this point in Section 3.7.

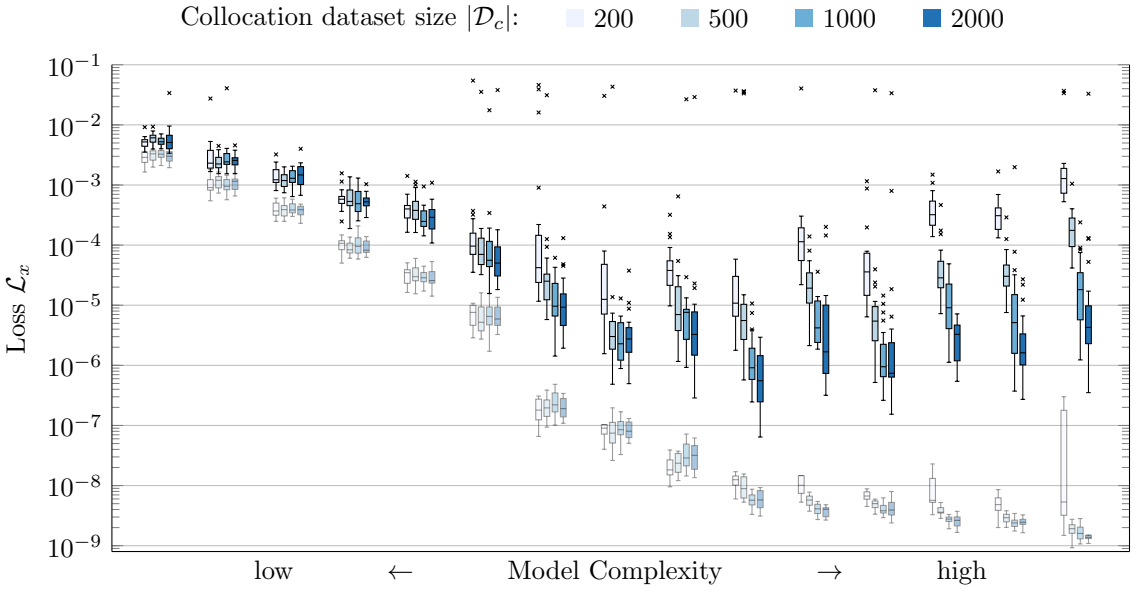


Figure 3.15: Impact of increasing number of collocation points (200, 500, 1000, 2000) on training (light colouring) and testing loss.

"upstream", i.e., on the states earlier in time. When we do not have any simulated data points, the only "anchor" point, where we know the state value, is the initial condition $\boldsymbol{x}_0$ at $t = 0$. This means, that we can only predict accurately at, e.g., 1 s, if we have also accurately predicted the flow Φ up to this point in time. If we have large errors close to the initial condition, this error will affect the entire trajectory or flow that follows.

Temporal causality - the need to start from the initial condition

The first imperative is therefore to accurately approximate the value at the initial condition to respect the temporal causality. Here, the design of the NN architecture provides an important tool. By using the formulation (3.31), we refer to it as *Residual A*, we place a hard constraint on the initial condition, hence, it is satisfied under all circumstances. Similarly, we can construct a NN with (3.32), that we call *Residual B*. It obeys also the first temporal derivative for the initial condition. To illustrate this, we show in Figure 3.16 the rotor angle δ for two trajectories at three epochs during a training process that is based entirely on collocation points. The black dashed line represents the true trajectory, the red one Residual A and the blue one Residual B. We also added a *Residual* NN (based on (3.30)) where no hard constraint is present. We can see that right from the start of the training (see Epoch 2), the initial condition is matched for Residual A (red) and Residual B (blue), and also the derivative for the latter. At epoch 10 the resulting trajectories begin to match the true trajectory more and more. For the trajectory starting at about 100 deg, a mismatch to the true trajectory occurs early on. Afterwards, the shape is approximately correct, but offset. This can only be resolved by reducing the early error, but after more training, i.e., in epoch 500, this is the case. To accelerate this process, the authors in [80] motivate an adapted training routine to respect the temporal causality. They introduce a weighting scheme that ensures, that the trajectory around the initial condition is fitted well first, before attempting to fit large time steps. The trajectory stemming from the Residual NN does not fit the true trajectory at all, even though it might obey the governing ODEs; we are missing the anchor point at the initial

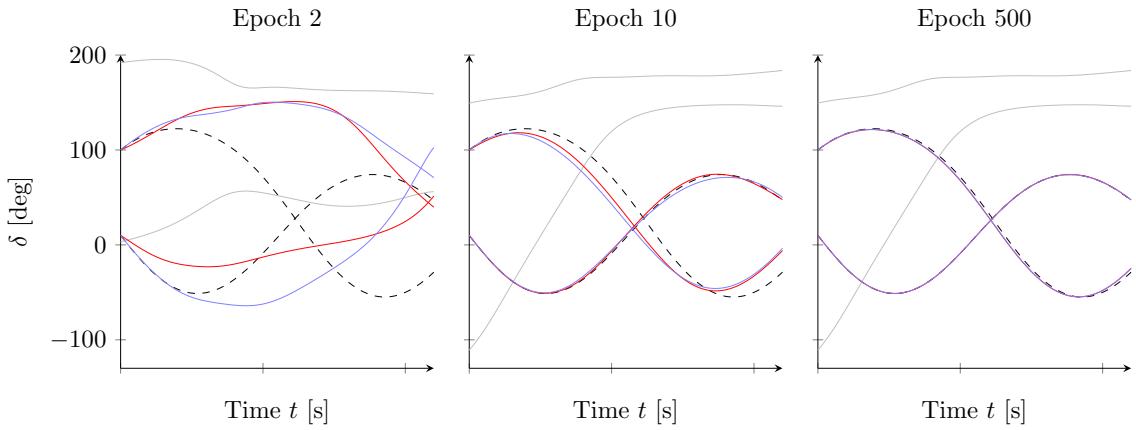


Figure 3.16: Two trajectories at different epochs of entirely collocation based training. The colours show signify different NN architectures, namely Residual (grey), Residual A (red), Residual B (blue).

condition[16]. Does it make a difference if we use a ResidualA or a ResidualB NN? We will first discuss another architecture-related possibility to train PINNs without a simulation dataset, namely IRK-PINNs, before coming back to this question.

IRK-PINNs

As alluded to earlier on, we need sufficiently many collocation points if we want to predict larger time step due to the need to propagate the solution from the initial condition onwards. The advantage of IRK schemes is that they can accurately predict long time steps and it "only" requires solving the associated system of implicit equations. By incorporating this scheme in NNs, i.e., constructing IRK-PINNs (see (3.33)), we can circumvent the need to resolve the entire trajectory upstream. The loss for each collocation point can be minimised independent of the others. Thanks to the loss $\mathcal{L}_{IRK}$, IRK-PINNs can also be trained as a flow approximator without any simulated data points. As described in Section 3.5.3, we can additionally use the physics loss $\mathcal{L}_c$ from PINNs, we refer to this procedure as IRK-PINN+. We will explore their effectiveness in the following paragraph.

Test loss results

We compare the different options to train a NN-based flow approximator $\hat{\Phi}$ without a single simulation, i.e., when only using collocation points. We fixed the model complexity to a medium level and will vary the size of the collocation dataset. The tested architectures are the Residual A and Residual B formulation as well as IRK-PINNs and IRK-PINN+. For the IRK-PINN and IRK-PINN+, we test different number of IRK stages ν. Figure 3.17 shows the resulting test loss. We observe that more collocation points help reducing the test loss. However, for the IRK-PINNs and IRK-PINN+, the test loss does not improve further from 1000 data points on; the models do not have sufficient representation capacity. For the Residual A and Residual B architecture, this point seems not to be reached yet for 2000 data points. If we compare the loss with the levels achieved with the same model complexity (see the middle column in Figure 3.6) beforehand, we

[16] By supplying a dataset only consisting of initial conditions, we could also make the Residual NN fit. However, in that case it is only a soft constraint and we have to penalise its violation by including $\mathcal{L}_x$ again in the loss function. This, however, then requires balancing $\mathcal{L}_c$ and $\mathcal{L}_x$. The hard constraint is the much more effective solution.

can see a notable disparity. When using simulated data points a loss of around 10^{-7} should be achievable, from which we can conclude that with more points, the Residual A and Residual B PINNs should achieve these levels. To give a perspective on the performance $\mathcal{P}$, the maximum error of the rotor angle lies consistently slightly above 10 deg for a test loss of 10^{-4}. The performance of the IRK-based models yield approximately this performance in the given setup. When reducing the test loss by a factor of 100, which is roughly the case for the Residual A and Residual B models, the maximum error can be reduced by a factor of about 10, i.e., to low single digit values for the rotor angle error.

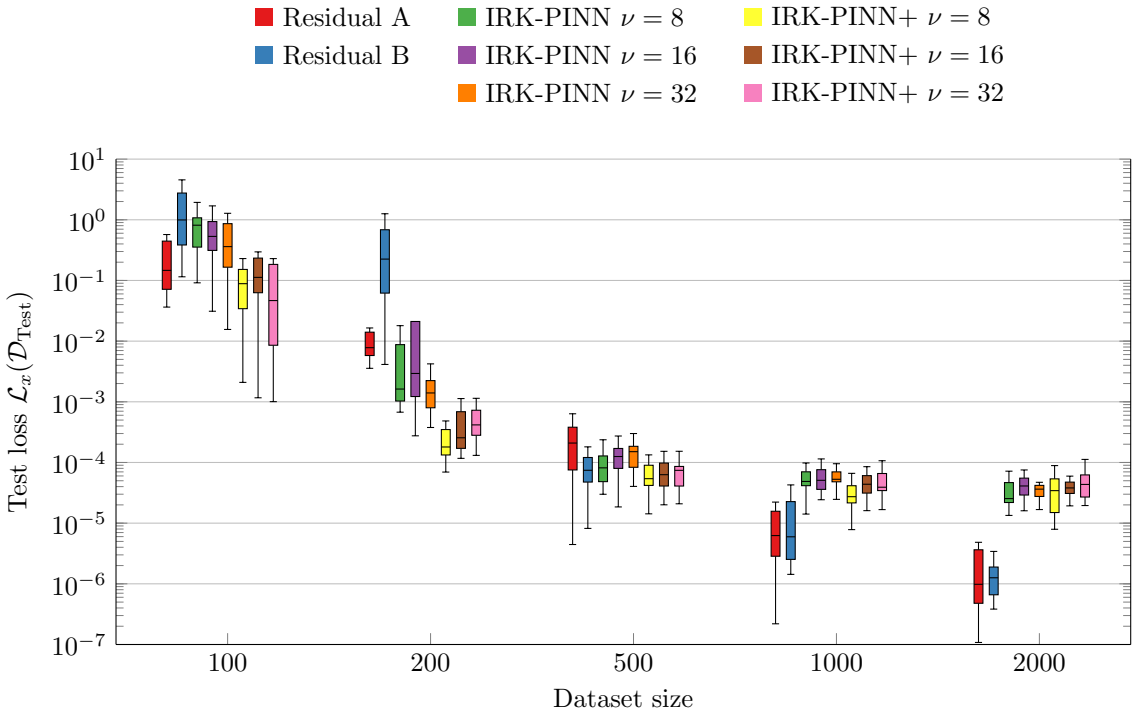


Figure 3.17: Test loss for PINNs trained purely with different numbers of collocation points (x-axis).

Ideally, we will only need small dataset sizes to achieve well-performing models. For collocation points, this is not driven by the effort to create the dataset – the collocation dataset only requires sampling and no simulation, thereby, it is essentially for free – but by the consideration of the BVTO. The need for few points to achieve sufficient accuracy implies good inductive biases. Their effectiveness is not too critical for the two-dimensional state, however, for a higher dimensional dynamical system, i.e., a system with more states, "efficiency" matters a lot more, as we encounter the *curse of dimensionality* (we will treat it in Section 6.5). The IRK setups seem to generalise a little better for the very small dataset cases, the reason lies most likely in the fact that they do not rely on accurate predictions upstream. Furthermore, the use of IRK-PINN+ leads to slightly lower losses than the IRK-PINNs due to the additional regularisation. The number of stages ν does not seem to greatly matter. The existing differences vanish as soon as the representation capacity becomes limiting. While the output layer of IRK-based PINNs has a lot more neurons (the stages times the number of states $\nu \cdot n$) than the Residual A and Residual B architecture (only n), the hidden layers before are of the same size. Loosely speaking, we cannot pass sufficiently differentiated information through the hidden layers, for predicting accurate IRK stages. Adjusting

the width of the hidden layers is an option, but it also makes the training more expensive. The two residual architectures yield similar performance, providing no argument (in this setup) for one or the other.

3.6.3 The loss as a distribution - the effects of the dataset design

So far, we mostly looked at the loss $\mathcal{L}_x$ to analyse model performance. Whenever distributions were involved, they were concerned with how the loss $\mathcal{L}_x$ behaves under different initialisations. This means that we only considered the mean of the loss values per point $\ell^{(i)}$

$$\mathcal{L}_x = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \underbrace{\left\| \hat{\boldsymbol{x}}^{(i)} - \boldsymbol{x}^{(i)} \right\|_{T,2}^2}_{\ell^{(i)}}. \tag{3.35}$$

As we motivated in Section 3.3, optimising the mean *might* also lead to a good performance using the metric $\mathcal{P}$, here the maximum absolute error, but it is not guaranteed. By analysing the distribution of the point-wise loss $\ell^{(i)}$ instead, we obtain a much more detailed picture. To do so, we evaluate a test dataset $\mathcal{D}_{\text{Test}}$ and then rank the values of $\ell^{(i)}$. By computing the percentiles of this ranking of ℓ, i.e., finding the value of ℓ_X such that X% of the values are smaller than ℓ_X, we obtain a detailed and comparable insight into the performance of the learned model. The 100-th percentile corresponds to the maximum value and the 50-th percentile to the median. We use this in-depth analysis to compare the dataset design options introduced in Section 3.5.2. It also lays the ground for a similar analysis in Section 3.6.4.

Sampling time

We have two sampling strategies to test the distribution of the input t, i.e., $p(t)$, a uniform and a log-uniform distribution. Importantly, training and test dataset to do not need to share the same distribution. In fact, this analysis aims exactly at understanding the effect of $p(t)$ used for the training dataset by testing it on other possible choices of $p(t)$ for the test dataset. We train 20 vanilla NNs on each of the distributions and then test each of the resulting models on the two test datasets. After computing the percentiles, we summarise the 20 resulting percentile curves in Figure 3.18. The shaded area indicates the region between the largest and lowest percentile value and the solid line the median across the 20 curves. The colours indicate the training dataset that was used and the two panels distinguish the test dataset.

If the sampling distribution $p(t)$ was irrelevant for generalisation, we would expect that the model trained on the same distribution as the test dataset should perform best. For most of the percentiles this is true; the red curve is lower than the blue one in the left panel and vice-versa in the right panel. This does not necessarily hold in the tails of the distribution, i.e., for low and high percentiles. The low percentiles, i.e., small values of $\ell^{(i)}$, are not of interest, but the percentiles close to 100 are extremely important as they relate to the largest error, and, hence, will determine the performance $\mathcal{P}$ of the approximator. In either test dataset, the models trained with a log-uniform sampling (blue) show significantly larger errors in the tail – this indicates that the uniform sampling leads to better generalisation when the maximum loss $\ell^{(i)}$ is concerned.

The other interesting observation is that the range of the percentile curves originating from the log-uniform sampling is substantially larger than for the uniform sampling. This indicates the dependence on the realisation of the dataset sampling; there can be “favourable” and “unfavourable”

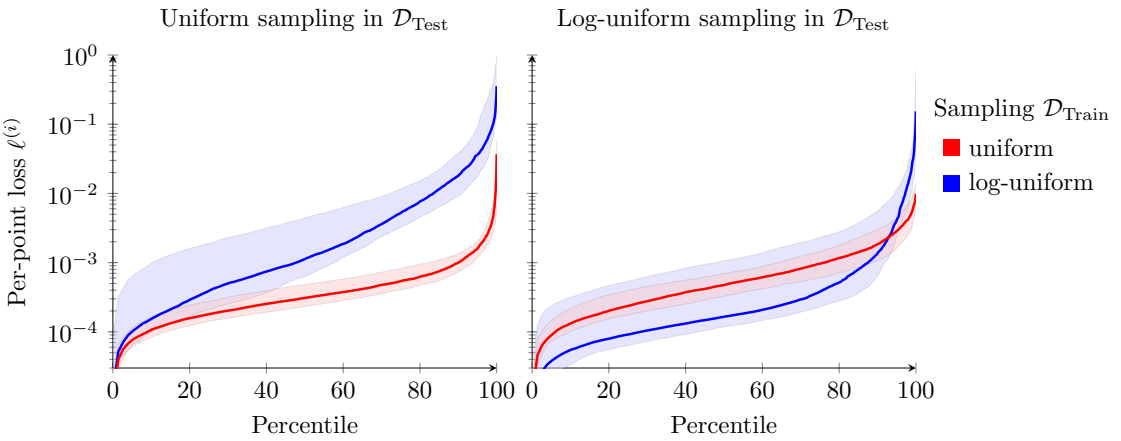


Figure 3.18: Distribution of the error percentiles for different sampling strategies for $p(t)$ in the training (colour) and testing (figure columns) datasets.

training datasets in terms of how well the model performs on the test dataset. A wide band means that the training outcome has a high dependency on how favourable the training dataset is. A narrow band is more robust in this regard and therefore preferred (if the performance overall is acceptable).

Sampling initial conditions

We repeat this kind of experiment also for the distribution of the initial condition, i.e., for $p(\boldsymbol{x}_0)$. Here we test three distribution, termed "radial", "annular", and "square" (see their definitions in Section 3.5.2). The results are shown in Figure 3.19 and the construction is identical to Figure 3.18. The trends observed in the previous paragraph largely hold; the tails are the critical, but also

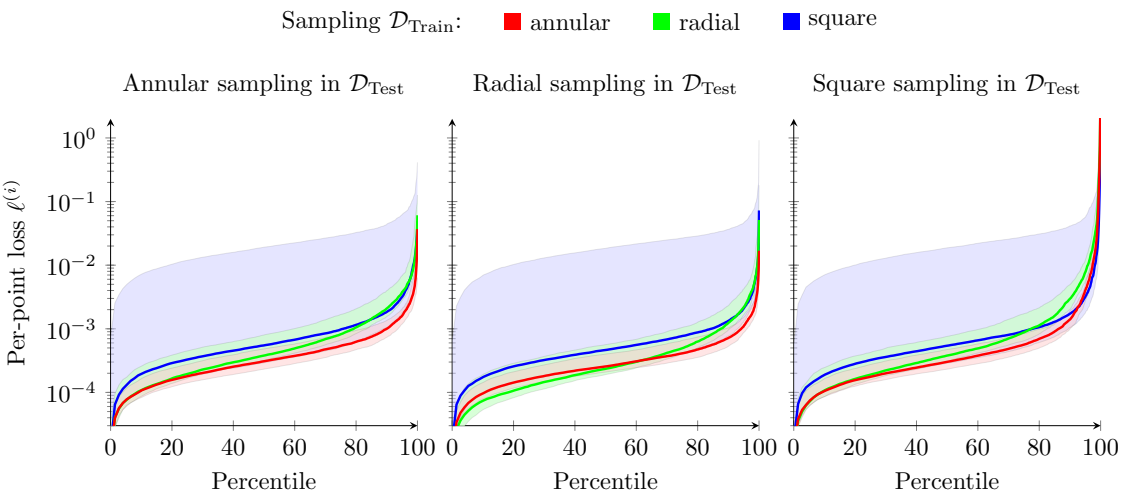


Figure 3.19: Error quantiles for different sampling strategies for $p(\boldsymbol{x}_0)$ in the training (colour) and testing (figure columns) datasets.

interesting regions. We want to note two important points: First, the models trained with the square sampling (blue) show a very large range of percentiles curves, even when applied to the test dataset with the square sampling. The median is also notably higher than for the other sampling

strategies. This indicates poor generalisation behaviour, and we can explain it when considering the phase space as we will do in the next section. Closely linked is the second observation. It concerns the maximum losses that we observe for the test dataset based on the squared sampling. This is the only place, where training with a dataset from the square sampling has a slight advantage over the other sampling strategies. At the same time, these losses are much higher than on the other test sampling strategies.

Preferred sampling strategy

From these analyses, the preferred sampling strategy seems to be a uniform sampling of time and an annular sampling of the initial condition. They show lower dependency on the randomness of the parameter initialisation and the dataset realisation, both are highly desirable for good generalisation. However, as we, ultimately, care about the maximum errors (according to $\mathcal{P}$), these mentioned properties are not sufficient. To analyse *where* the maximum errors occur, we continue in the next section with forming conditional errors on the input values t and $r(\boldsymbol{x}_0)$.

3.6.4 Shaping conditional error distribution

In this section, we compute *conditional* loss distributions $p(\ell|t)$ and $p(\ell|r(\boldsymbol{x}_0))$ given time t or the radius of the initial condition $r(\boldsymbol{x}_0)$. Our aim is to not only know the error distribution based on percentiles as in the previous section, but also identify *where* in the input domain we can expect small or large loss values. To do so we partition the test dataset evaluated on a single trained model into small segments based on t or $r(\boldsymbol{x}_0)$ and then compute the minimum, maximum, and median loss of the points within a segment.

Loss conditional on time

To begin, we analyse the dependency of the loss on the time step size, shown in Figure 3.20. We

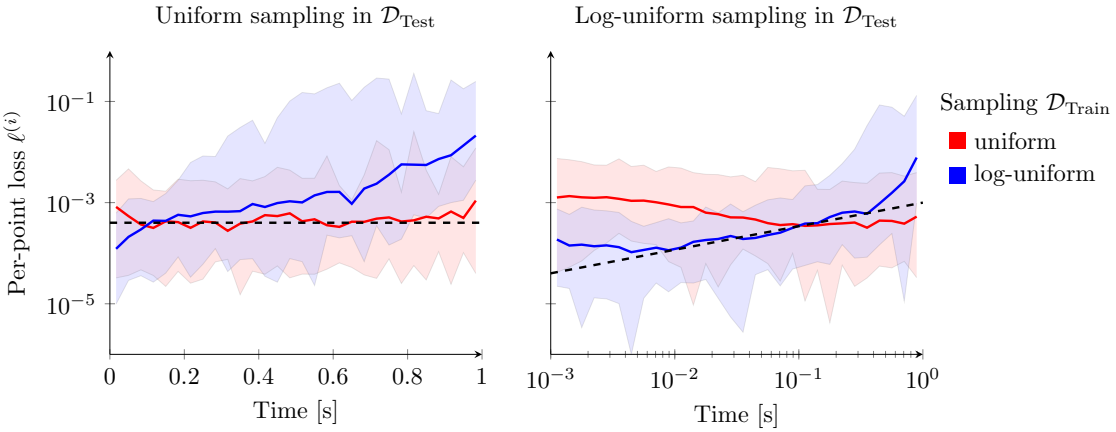


Figure 3.20: Conditional loss distribution $p(\ell|t)$ with uniform (red) and log-uniform (blue) sampling. The solid line represents the median and the shaded area the range between maximum and minimum loss. The figure columns correspond to the used test dataset.

observe in the left panel that the uniform sampling in time leads to an approximately uniform conditional loss distribution. Neither the mean nor the extreme values show a clear trend with respect to the input t. In contrast, the log-uniform sampling has a clear dependency on where

we evaluate it. Very small time steps are well approximated, below 0.1 s even better than for the uniform sampling. However, large time steps incur significant losses. This characteristic has a reminiscence of the dependency of the error on the time step size seen in numerical approximation schemes, see Section 3.2. If we look at the right panel (note the a logarithmic x-axis) that is evaluated on the test dataset from the log-uniform distribution, we can get a more detailed picture. In the region between 0.01 and 0.2, the blue curve and the shaded area are approximately linear – the slope of the dashed line would correspond to the order p $\mathcal{O}(h^p)$. Outside this region, we can observe the effect of being close to the input domain boundary, i.e., the minimum and maximum values of t. When training a model, the generalisation at the boundary is usually worse than in the middle of the domain. The reason is that there are fewer points that determine the fit, and hence overfitting and worse generalisation are the outcomes. Therefore, the conditional loss distribution does not follow the dashed trend-line any more. We can observe a similar but less extreme effect in the left panel for the uniform distribution (red). It slightly deviates from the dashed line towards higher losses close to the boundary. In the right panel, this becomes a clearer, where small time step have higher errors than for most of the remaining domain.

Loss conditional on the radius of the initial condition $p(\ell|r(\boldsymbol{x}_0))$

Figure 3.21 shows the corresponding results of the loss conditional on the radius of the initial condition. For the annular sampling, that aims to achieve an equal coverage of the unit disk in

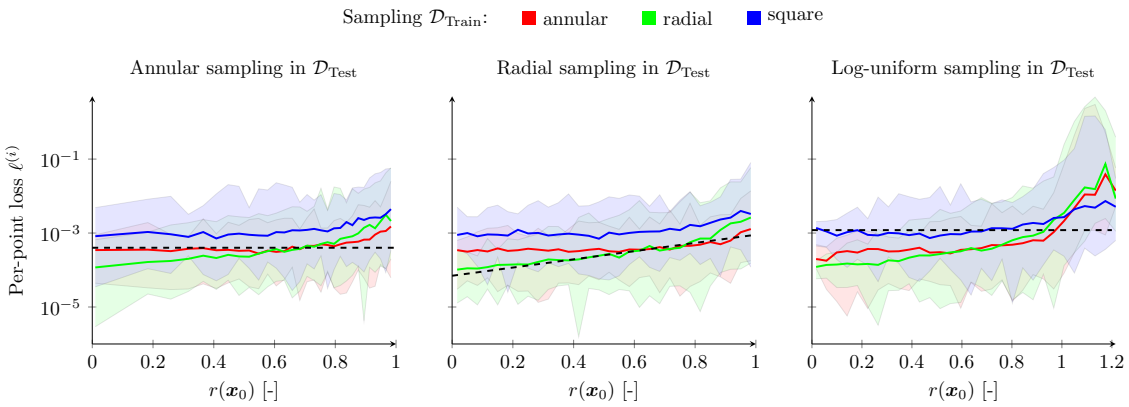


Figure 3.21: Conditional loss distribution $p(\ell|r(\boldsymbol{x}_0))$ with annular (red), radial (green) and square (blue) sampling. The solid line represents the median and the shaded area the range between maximum and minimum loss.

the dimensionless state space, we obtain an error distribution that is nearly independent of the radius of the initial condition. The argument for the boundary now only holds for the upper value of $r(\boldsymbol{x}_0)$, as a value of $r = 0$ corresponds to the equilibrium point, i.e., it sits at the centre of the input domain. For the radial sampling, we observe a dependency on the radius, as there were more points with a small radius in the training dataset. This leads to a better generalisation for a small value of $r(\boldsymbol{x}_0)$, but worse when we approach the boundary of the input domain. The square sampling leads to an error distribution that is approximately independent of the radius up to a value of $r(\boldsymbol{x}_0) \approx 1$ as the training data points are equally spread over the area of the input domain. However, the maximum radius is not equal in all "directions" in the state space as shown in Figure 3.22. Therefore, the generalisation of the model trained with the square domain begins to worsen from $r \approx 0.8$ on. The other two sampling strategies maintain better accuracy up to $r \approx 1$

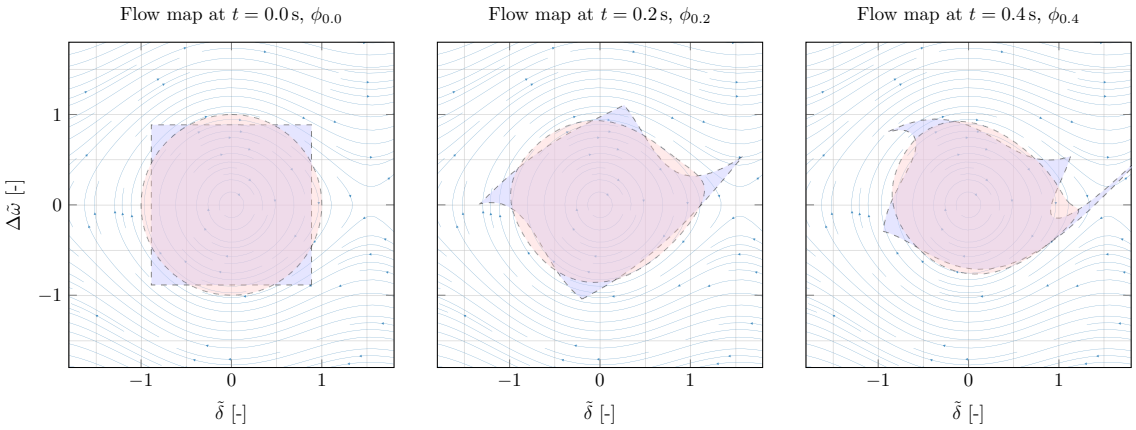


Figure 3.22: Flow maps for the different sampling domains $\mathcal{X}_0$ of the square (blue) and annular/radial sampling (red). Due to the non-linear dynamics the initially round and square domains get distorted over time. In case of the square sampling, the effect is greater as the maximum radius varies for $t = 0.0\,\mathrm{s}$.

(the boundary effect sets in), but then become even worse than the model trained with squared sampling. This drop in accuracy is expected as it concerns generalisation beyond the training domain. The overall higher loss for the squared sampling strategy can be attributed to the need to learn a more complicated function. Avoiding large errors for $r > 1$ comes at the expense of accepting higher error for $r < 1$. To be accurate though, this setup constitutes a slightly different task $\mathcal{T}$ as the domain $\mathcal{X}_0$ differs from the evaluation on the other two test datasets. As this task $\mathcal{T}$ seems to be more difficult, its less accurate solution is not surprising.

Implications for the choice of sampling strategy

These analyses lead to two conclusions. First, it is to be expected that the largest errors occur at the sampling boundary. If we evaluated the performance metric on a reduced input domain, e.g., only for $r < 0.9$, we obtain a better performance. This is turn means, that to improve performance for the original domain, we can extend the input domain for training, e.g., $r_{\max} = 1.2$ to avoid the boundary effect at $r = 1$. The same holds for the time step size t, extending to negative values of t is also possible. Second, by changing the distributions $p(t)$ and $p(r(\boldsymbol{x}_0))$, we can induce conditional error distributions. More points in a region of the input domain allow for more controlled fitting of the NN and hence better generalisation. However, it is preferable to incorporate these characteristics by hard constraints, e.g., the architecture of the NN. For the case of the time step size h, we can go back to Figure 3.3. The conditional error on the time step size h that shown, was achieved by the Residual B formulation (the other curve corresponds to the simple residual formulation). The conditional error will be dependent on the time step size as $\mathcal{O}(h^2)$ for small time step size. By using the architecture to ensure this numerical consistency, we can use the points in the dataset to focus on the predictions for larger time step sizes.

In terms of sampling the initial condition $\boldsymbol{x}_0$, the equal coverage of the input domain is desirable. At the same time, the "isotropy" properties, i.e., are there directional dependencies, seem to play a role. Here, once more, the introduction of the dimensionless state space and the definition of the radius r, proved to be of great value as it enables the analysis of a conditional error distribution with respect to the initial condition.

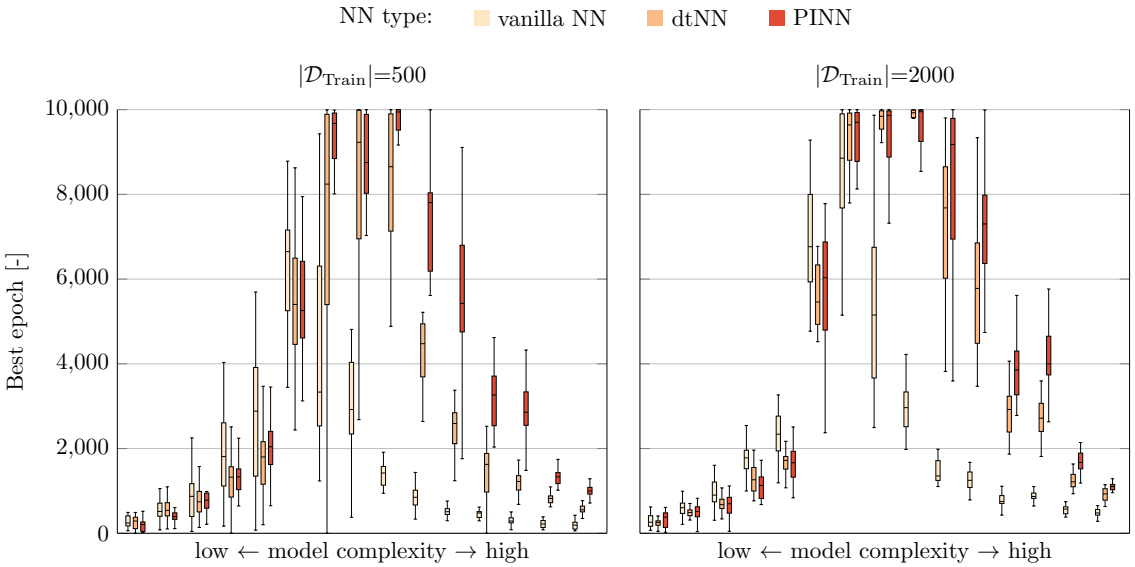


Figure 3.23: Best epoch E^* for NNs, dtNNs, and PINNs (coloured), for different dataset sizes (panels) and model complexities (x-axis).

3.6.5 Impacts on the training duration

So far, we have discussed many ways to influence the BVTO and the involved error characteristics from the perspective of approximation accuracy, i.e., the loss $\mathcal{L}_x$ and the performance $\mathcal{P}$ on a task $\mathcal{T}$. However, when it comes to the decision which combinations of model complexity, dataset size, and additional inductive biases to choose, the time required for training must also be taken into account. We want to stress that the evaluation time is mostly unaffected by these choices, except for the size and structure of the hypothesis class h_θ which are the main drivers of the evaluation time (see Section 3.2). Similar to the difficulty of quantifying the speed of different predictor schemes (see Section 3.2), the analysis of the training duration has many influencing factors. To disentangle those, we consider the number of epochs E^* until the best performance is reached and the time required per epoch C_E. The total training time is proportional to each of them, i.e., $C_T = \mathcal{O}(E^* \cdot C_E)$. The following analyses investigate the influencing factors on each of them.

Best epoch - early stopping

As we elaborated earlier on, monitoring the loss on a validation dataset $\mathcal{L}_x(\mathcal{D}_{\text{Validation}})$ is important to identify the onset of overfitting the model to the training dataset $\mathcal{D}_{\text{Train}}$. Under the premise that the validation loss provides a reliable estimate of the generalisation[17], the epoch with the lowest validation loss would be ideal to stop, i.e., this defines E^*. Continued training would only add training cost at no accuracy gains. In Figure 3.23, we plot E^* against the model complexity on the x-axis. The colours of the boxplots (we test 20 training runs) indicate the NN type, i.e., vanilla NN, dtNN, PINN, and the two panels show different training dataset sizes. We observe that the medium complexity models train for the most epochs, they would even continue above the set upper limit of 10000 epochs. Low complexity models can be fit very quickly due to fewer parameters and quickly reaching the bias-dominated performance. The highly complex models on the other hand, have enough representation capacity to fit the training data within relatively

[17] For the presented setups this was usually the case, very small datasets form an exception.

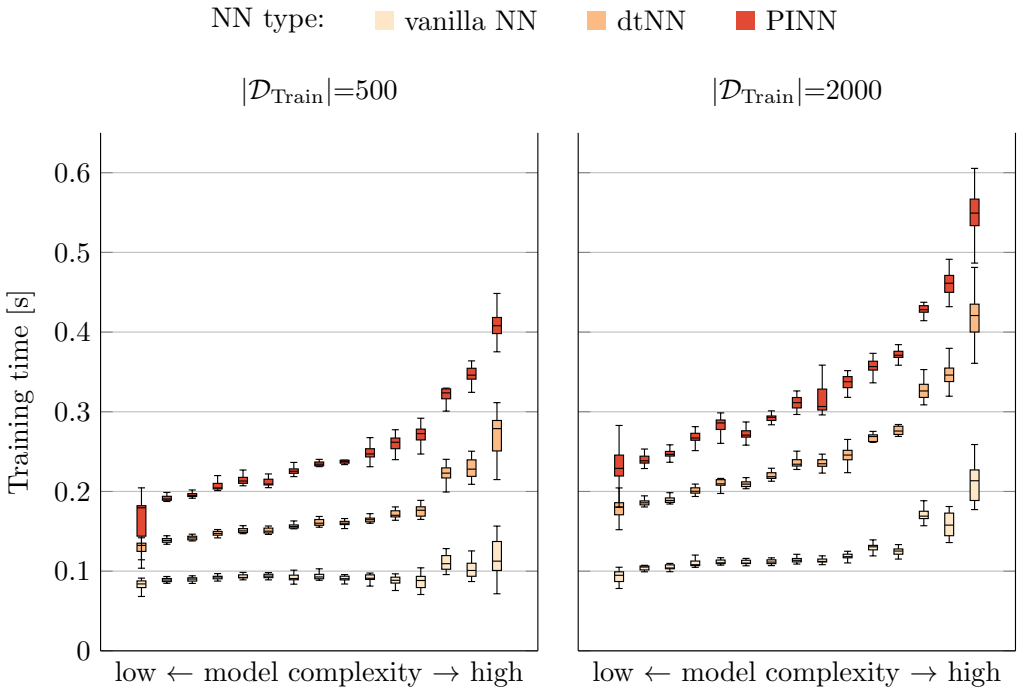


Figure 3.24: Training time per epoch for NNs, dtNNs, and PINNs (coloured), for different dataset sizes (panels) and model complexities (x-axis).

few epochs, additional epochs then lead to overfitting. The models with medium complexity can improve for many more epochs, however, when factoring in the trends in the achieved accuracy, e.g., Figures 3.7 and 3.15, these additional epochs do not always lead to a significantly improved accuracy. In particular for larger datasets, the gains are relatively small but require many more epochs. The improvements in accuracy from including dt-loss ($\mathcal{L}_{lhs}$, $\mathcal{L}_{rhs}$) and the physics-based loss $\mathcal{L}_c$ also require training for more epochs, except for low complexity models. The settings of the L-BFGS optimiser do also play a role[18], however, their analysis does not provide further useful insights, unless going into great detail of the optimisation landscape and the optimiser itself, which was out of the scope of this thesis.

Time per epoch

The other important driver for the training duration is how much time each epoch requires. As in the previous section, we plot the time for the three NN types over increasing model complexities and different dataset sizes. The results are shown in Figure 3.24. The two general and unsurprising trends are that the time per epoch increases both with increasing model complexity and larger datasets.

The trends, that we observe, align with the results presented earlier on speed. We expect larger datasets and more complex models to require more time per epoch. However, the increased dataset size has a sub-linear impact, i.e., a four-fold from 500 to 2000 data points does not lead to four times longer training time. The reason presumably lies in the dependency of the evaluation speed

[18]Each epoch consists of multiple internal steps of the optimiser. The number of steps will influence the improvement that we can achieve within each epoch. However, how many steps we ideally take is very dependent on the loss landscape and the estimation of its Hessian. A dedicated analysis of the optimiser and its settings might yield general insights, but this was beyond the scope of this work.

on the number of points as discussed in Section 3.2. The evaluation of additional loss terms requires also additional time, therefore, dtNNs require about twice the time of a regular NN. This is due to the additional pass needed for calculating the time derivative $\frac{d}{dt}\hat{\boldsymbol{x}}$. The extra time for PINNs, i.e., on top of dtNNs, depends on the number of collocation points, here 500, in comparison to the dataset size. Therefore, the difference between dtNNs and PINNs reduces here with increasing dataset sizes. The larger number of parameters for models with higher complexity leads to a longer training time per epoch. For second-order optimisers, such as the used L-BFGS optimiser, this is highly relevant due to the calculation or approximation of the Hessian of the loss with respect to the parameters.

As mentioned before, the absolute numbers are hard to compare and effects like limited memory, a concern with the L-BFGS optimiser, can occur. Furthermore, the settings of the L-BFGS optimiser (not altered in these setups), such as the internal number of steps and the history size for approximating the Hessian, play an important role for the training time per epoch.

Optimising for training speed

In this chapter, we focused primarily on achieving high accuracy – the necessary condition for NNs to be an attractive alternative to conventional numerical schemes. The investigation of accelerating their training will be an important step in future work on PINNs as flow approximators.

3.7 Characterising the learning task - related works

In the following, we want to give some more context to the presented results, both conceptually and with respect to the literature. To this end, we begin by discussing the differences between discrete and continuous approximators, i.e., flow map and flow approximators, and schemes that rely on a direct prediction or a time-stepping scheme. Lastly, PINNs have been largely studied in the context of PDEs rather than ODEs, hence, we want to provide a perspective on the implications when using PINNs for ODEs.

The topic of PINNs has seen an enormous development over the past few years. The authors in [82] state that the literature of this emerging field is "still disorganized, with a somewhat unstable nomenclature" which matches our experience. Therefore, we refrain from discussing approaches in all their detail, several comprehensive reviews have already been written [28], [30], [83], [84], but try to organise concepts instead. In our view, a lot hinges around having clarity on the formulation of the problem that we are trying to solve, as this allows the classification and comparison of methods. Throughout this chapter, taking the viewpoint from the stable and established field of numerics proved to be helpful, hence we will continue to do so. There are many other valuable viewpoints that are worth mentioning without going into great detail. These connections can hopefully help obtaining more clarity on the topic.

3.7.1 Discrete or continuous predictor? Direct or time-stepping method?

When setting up the learning task $\mathcal{T}$ in Section 3.3, we have skipped some decisions that ought to be made beforehand. Let us consider the case that we want to predict the trajectory from the current state, i.e., the initial condition $\boldsymbol{x}_0 = \boldsymbol{x}(t)$, up to time $t = t_0 + \Delta t$. The two vertical lines in Figure 3.25 shall illustrate that time interval. In such a setting, we can decide if we predict the entire interval at once or split it into smaller time steps of size h. The authors in [85] refer to it as

direct and *time-stepper* methods, for the latter we will use the term *time-stepping* method. The other important distinction is, whether we predict the output state in a *continuous* or a *discrete* fashion – it is the question whether h is used as an input to the predictor (continuous) or if it is fixed (discrete). For the former, we can adjust the size of h throughout the simulation of the time-interval, for the latter we cannot. In the language of numerics and flows, the continuous predictor approximates a flow Φ, a discrete predictor approximates a specific flow map ϕ_t.

Options to predict trajectories

If we are only interested in a single trajectory $\boldsymbol{x}(t; \boldsymbol{x}_0)$, the direct predictor does not need to approximate the entire flow Φ which maps time and state to a new state $\mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n$; approximating the particular trajectory which maps time to a state, i.e., $\mathbb{R} \mapsto \mathbb{R}^n$ is sufficient. The trajectory of interest is a subset of the flow. As soon as we introduce a time-stepping scheme, we inevitably encounter different initial conditions at the beginning of the time steps (unless the system is in its equilibrium). If we want to use the same predictor for all time steps, this requires us to approximate trajectories with different initial conditions, i.e., approximate a flow $\hat{\Phi}$ or a flow map $\hat{\phi}_t$. The simulation then only requires the repeated application of the flow or flow map, i.e., $\hat{\Phi}(\cdot, \Delta t) = \hat{\Phi}(\cdot, h) \circ \hat{\Phi}(\cdot, h) \circ \cdots \circ \hat{\Phi}(\cdot, h)$ or $\hat{\phi}_{\Delta t} = \hat{\phi}_h \circ \hat{\phi}_h \circ \cdots \circ \hat{\phi}_h$. When we use the continuous flow approximation $\hat{\Phi}$, we can also alter h over the time interval. However, as we use approximations, the properties of flows and flow maps as described in Section 2.2 do not hold any more. For instance, the choice of the time step size matters as usually $\hat{\phi}_{2h} \neq \hat{\phi}_h \circ \hat{\phi}_h$. If we reduce the size

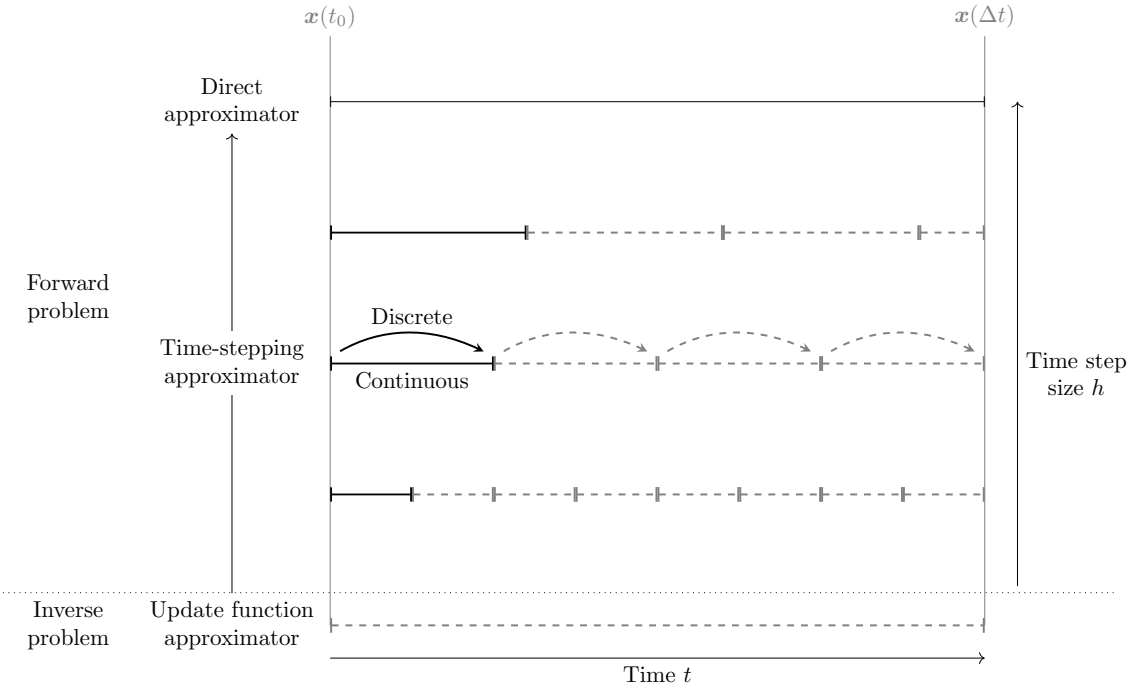


Figure 3.25: Schematic classification of ODE related approximators. They can be classified by their time step size h, ranging from a direct approximator to time-stepping approximators for the forward problem. In the limit of $h \to 0$ we obtain an approximator of the update function, i.e., a solution to the inverse problem. For the forward problem, one can furthermore distinguish between discrete and continuous approximators that correspond to flow map and flow approximators $\hat{\phi}_t$ and $\hat{\Phi}$.

of h towards 0, we obtain in the limit the update function $\boldsymbol{f}$. At this point, we move from the forward problem, i.e, approximating the trajectory, to the inverse problem, i.e., approximating $\boldsymbol{f}$ – the output from the inverse problem is not any more a trajectory, but just the update vector. With this categorisation in mind, we want to look at a few predictor schemes, mostly learning-based.

RK schemes

The RK schemes, explicit and implicit, both fall into the category of flow approximators $\hat{\Phi}$, i.e., continuous methods. As the accuracy is often limited to relatively small time step sizes h, we pre-dominantly encounter them in time-stepping schemes. When applying RK schemes they may seem "discrete", as we only obtain a single state upon querying them, however, the function that they describe is well-defined for a continuum of time step sizes, i.e., they are continuous methods. This also holds for IRK schemes, even though we first need to solve a system of implicit equations. As long as the implicit function theorem can be applied, the solution to this system of implicit equations will describe, at least locally, a continuous function.

PINNs

PINNs, as we have presented them, fall into the same category as RK-schemes, i.e., continuous flow approximators $\hat{\Phi}$. The IRK-PINNs as introduced in [26] form the exception, as they are approximating flow maps $\hat{\phi}_t$. With our proposed adaptation to include the time step size h as an input, we recover the continuous flow approximator type also for IRK-PINNs.

RNNs, LSTMs - applying time-stepping internally

Recurrent Neural Networks (RNNs) summarise a class of models that build an internal representation, a hidden state, which then is propagated through a sequence of operations [59]. By using the same parametrisation of the operation, RNNs can achieve improved generalisation as the operation needs to be applicable through the entire sequence. The authors in [49, Ch. 10] provide an introduction to this sequence-based modelling approach. A special form of RNNs are Long Short-Term Memory (LSTM) NNs [86], that alleviated a practical problem in training RNNs. It relates to the need to retain some information throughout the entire sequence while others can be "forgotten", the used structure to achieve such behaviour are called *gates*.

When viewed from a dynamical system perspective, as in [87], [88], RNNs are closely linked to a discrete time-stepping scheme. In fact, each step of the sequence evaluation can be seen as a flow map approximation $\hat{\phi}_t$. As the propagated hidden state usually does not equal the state of the physical system $\boldsymbol{x}$, the computations internal to the RNN become a flow map to an ODE of the hidden state. The system state $\boldsymbol{x}$ will be recovered based on a transformation of the hidden state. The step size h cannot be adjusted in this architecture, and it is also not an input.

Operator learning and Koopman operators

Mathematically speaking, the temporal integration can also be seen as an operator that maps between two function spaces. The flow Φ defines such an operator. By learning this integration operator in which the initial conditions, control inputs, and system parameters can be parametrised, we obtain a flow approximation $\hat{\Phi}$. The idea to learn operators was first introduced as DeepONets in [89] and led to several interesting works such as Fourier Neural Operators [90], Physics-Informed DeepONets [91], Physics-Informed Neural Operators (PINO) [92], and Neural Operator learning

[93]. When comparing this approach to the presented view of PINNs as flow approximators, they differ on the level of the hypothesis class. It is proven that operators can be approximated with NNs [94] and works like [95], [96] attempt to identify error bounds and convergence guarantees. These questions are also asked for PINNs, e.g., in [82], [97]. By moving from approximating a single trajectory to approximating a flow, it becomes apparent, that we effectively apply a form form of operator learning with PINNs.

The study of operators is not exclusive to Machine Learning (ML), the so-called Koopman operator [98] – see a comprehensive review in [99] – has a long history in studying non-linear dynamical systems. The operator is linear, which means that applying it to non-linear dynamical systems allows us to then treat these systems like linear dynamical systems. The crux lies in the fact, that the Koopman-operator is an infinite-dimensional operator; therefore, we need to find tractable finite-dimensional representations of the operator. Learning representations from data (experience) is what ML and data-driven methods excel at, hence, linking these concepts seems natural.

NeuralODEs, HNNs, LNNs - approximating $\boldsymbol{f}$

A number of approaches have been proposed to approximate $\boldsymbol{f}$ based on observed data. Examples are Neural Ordinary Differential Equations (NeuralODEs) [100], Hamiltonian Neural Networks (HNNs) [32] and Lagrangian Neural Network (LNNs) [33]. The goal is to describe the update function $\boldsymbol{f}$ without specifying the functional form of the ODE; it is the *inverse* problem which we will discuss in Chapter 4. The learned NN does not represent a flow or flow map approximation, hence, the simulation of a trajectory will require a separate ODE solver. HNNs and LNNs formulate the learning problem such that the conservation of energy in Hamiltonian or Lagrangian coordinates is guaranteed. It is a good illustration of how an inductive bias founded on domain knowledge can be included in the NN architecture and thereby boost performance.

3.7.2 Specifying the task

This brief overview of prominent modelling approaches shows the range of models from which modellers can choose. Specifying the task in more detail will determine which approaches are suitable and which are not. Hence, we want to give a few considerations on what makes tasks difficult or easy (see Sections 3.3 and 3.4)

The difficulty of the learning task $\mathcal{T}$ for continuous schemes depends on the maximum time step size $h_{\max}$ that we choose. The larger h is, the more "volume" of the flow we have to predict and non-linearities might become more pronounced. Therefore, learning a direct solution becomes increasingly difficult with longer time intervals – a problem termed "long-time integration" which PINNs are not ideally suited for and where operator learning might help [91]. In this context, temporal causality that affects the training of PINNs, as [80] describes, poses a challenge for large $h_{\max}$.

For discrete schemes, the time step size h does not directly affect the learning difficulty, it rather depends on how well the non-linearity can be captured. A more difficult decision is the choice of the time step size itself. As [101] exposes, learning continuous systems with inherently discrete models requires additional attention to phenomena of overfitting to certain discrete cases, i.e., to the time step size used for training.

The choice of the optimal maximum time step size will eventually come down to the global approximation error. The global approximation error depends on the local approximation error and

the number of time steps as the local approximation error can compound. For continuous numerical schemes, one can analyse the resulting order of a scheme. Under conditions of Lipschitz continuity and convergence of the approximator, the global approximator error scales with $\mathcal{O}(\frac{1}{h}h^{p+1}) = \mathcal{O}(h^p)$ where p is the order of a scheme, see, e.g., [11], [12]. For RK4 this means that the global approximation error scales as $\mathcal{O}(h^4)$; we do not know its absolute magnitude though. In Section 3.6.4 we analysed error characteristics for PINNs. As there can be regions where the local approximation error scales as $\mathcal{O}(1)$, i.e., it is independent of the time step size, fewer but larger time steps can reduce the global approximation error. However, learning a larger time step size will also make the learning task $\mathcal{T}$ harder, which in the worst case can lead to larger local approximation errors. For discrete approximators the similar logic of aiming for fewer but larger time step sizes can be made as long as the local approximation errors remain low.

The input domain for the initial condition $\mathcal{X}_0$ plays an important role for both types of schemes, continuous and discrete ones. The flow or flow map approximator needs to cover the entire range of states $\mathcal{X}$ that can be encountered at the end of each time step as a potential input $\mathcal{X}_0$. In the limit of very small time steps the predictor needs to be trained on all points that are encountered during a trajectory. This will usually enlarge the set $\mathcal{X}_0$ significantly, making the task harder. Contracting systems form an exception as the set of reached states $\mathcal{X}$ will shrink with increasing time. In dynamical system theory, the study of how these sets evolve and depend on the system is well established, e.g., see alpha and omega limit sets [53]. The need to train the approximator on the larger set of reached states $\mathcal{X}$, rather than only on the set of initial conditions that are eventually of interest, can be a drawback of time-stepping scheme. Direct approximators, in contrast, only need to cover the initial states that are of interest at t_0.

3.7.3 ODEs as a simpler case of PDEs?

The research on PINNs applied to PDEs has been thriving as recent reviews [28], [30], [83], [84] show. They cover all types of approaches from novel architectures to dataset sampling strategies to shaping the loss function. The study object, however, is often a set of standard of PDEs - Burger's, Schrödinger's, Poisson's equation. Therefore, we have to ask if learning solutions to ODEs is equivalent or how it differs.

An ODE can be written as a PDE

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{f}. \tag{3.36}$$

The only *dependent* variable of an ODE is time t and its solution is $\boldsymbol{x}(t)$. For PDEs, such as the heat conduction equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \tag{3.37}$$

there will be multiple dependent variables, here time t and a spatial coordinate x and the solution will be described as $u(t, x)$. To obtain a unique solution for $\boldsymbol{x}(t)$ or $u(t, x)$ we need to specify the boundary or initial conditions. These conditions can be parametrised with *independent* variables. System parameters also count as independent variables.

In this chapter, we used PINNs to predict a *flow* Φ instead of a single trajectories $\boldsymbol{x}$. It may appear that the initial condition $\boldsymbol{x}_0$ is thereby turned into a dependent variable. This is not the case as the governing ODE / PDE is

$$\frac{\partial \Phi}{\partial t} = \boldsymbol{f}. \tag{3.38}$$

For $\boldsymbol{x}_0$ to be a dependent variable we would require a term such as $\frac{\partial \Phi}{\partial \boldsymbol{x}_0}$. For the learned approximator $\hat{\Phi}$, one could evaluate $\frac{\partial \hat{\Phi}}{\partial \boldsymbol{x}_0}$, it would even carry the interesting meaning of a sensitivity of the resulting trajectory with respect to its initial condition. However, we cannot use this term in a PINN-like loss that penalises unphysical solutions.

Therefore, we have to distinguish between the dependent and independent variables when comparing ODEs and PDEs, e.g., in terms of their dimensionality. A 3D-PDE with t, x, y being dependent variables is different from an ODE with t as the dependent variable and $\boldsymbol{x}_0 \in \mathbb{R}^2$ as a two-dimensional independent variable – even though one might be tempted to say that it also has "three dimensions" in total. Solving ODEs with conventional numerical schemes is usually easier than solving PDEs as only one dependent variable, which requires discretisation, is present. Furthermore, we can solve ODEs for different values of the independent variables in parallel, e.g., simulating different trajectories.

This has also implications for the learning problem. For different values of independent variables, we can compute their respective solutions and create a dataset with exact solutions. PINNs, as usually applied to PDEs, do not alter independent variables, therefore, there is no possibility for generating a dataset with parts of the solution. For this reason, we were able to propose the use of the dt-loss term for ODEs. In a PDE-setting with no independent variables, evaluating a corresponding term is not possible. The proposal of operator learning, e.g., DeepONet [89], aims at including variations of independent variables for PDEs. For this reason, one might consider our approach of learning the flow Φ of an ODE as a form of operator learning. In our view, the study of ODEs as a particular case of PDEs should, nonetheless, receive special attention as many ODEs, that describe dynamical systems, do have a large number of independent variables: the initial condition $\boldsymbol{x}_0$, the vector of control inputs $\boldsymbol{u}$, and the system parameters $\boldsymbol{\lambda}$. This leads to a high-dimensional learning problem with its own challenges.

3.8 Concluding remarks

From this in-depth investigation of using PINNs for approximating the flow Φ of a dynamical system, we conclude that PINNs have very attractive numerical characteristics. As an explicit approximator, they are fast to evaluate, but in contrast to explicit RK schemes, they can also be accurate for long time step sizes – an unusual but highly desirable characteristic. By adjusting the architecture of the hypothesis class h_θ, the essential numerical feature of consistency can also be ensured. This describes a trained PINN though. It requires a learning process that, while conceptually simple, needs to navigate a large space of potential inductive biases to improve the learned outcome. These inductive biases are the key to good generalisation of the resulting models without requiring extensive simulated datasets. Changes to the architecture of the hypothesis class h_θ appeared to be the most efficient approach in this context. Furthermore, the definition of the suitable task $\mathcal{T}$ plays an important role in the learning procedure. This includes the selection of the maximum time step size that shall be learned, the definition of the range of initial states, the performance metric, and a suitable scaling of the system's states. We observed that the achievable accuracy in the training is one objective, however, it has to be balanced with the involved computational effort. Different setups can lead to similar performance but with widely ranging overall computational cost.

In the presented case, we can visualise the flow, consider states one-by-one instead of aggregated, and analyse errors in the various input dimensions to use visual insights to enhance understanding.

To a large extent, higher dimensional state spaces will prevent such intuition-building methods. Instead, we have to represent the quality of the model and its ability to generalise in few metrics, ideally a single number such that all tested models can be compared easily. To this end, theory building at the intersection of dynamical systems, numerical methods, statistics, and ML will be required - this chapter should have highlighted the need for all these disciplines.

In future work, we will, first and foremost, focus on extending the application of PINNs to a variety of dynamical systems that describe more complex dynamics and flows. This complexity can arise from a larger number of states, largely different time-scales (stiff ODEs), the variation of control inputs and system parameters, or discrete and algebraic variables in the description of the system dynamics. It will require to further formalise the concept of the dimensionless states and to develop a rigorous framework for the analysis of errors. By seeking a close connection to the emerging field of operator learning, advanced hypothesis classes will be developed. In this thesis, we did not investigate the influence of the optimiser and its setting in great detail. However, understanding the mechanics of the involved optimisation, which is closely related to the loss landscape, might lead to accelerating the learning and improving its outcome.

CHAPTER 4

PINNs for inverse problems

The first step before solving a forward problem, as in the previous chapter, is to solve the so-called inverse problem, also known as system identification: We try to find a model $\boldsymbol{f}_\lambda$ parametrised by $\boldsymbol{\lambda}$ that could have generated the data points observed in a given dataset $\mathcal{D}$, e.g., from experiments or operational data. This chapter will show how Physics-Informed Neural Networks (PINNs) offer a powerful while generic approach to this problem when Ordinary Differential Equations (ODEs) define the model. The use of Bayesian Neural Networks (BNNs) and Bayesian Physics-Informed Neural Networks (BPINNs) provides a probabilistic framework for solving such an inverse problem. At the same time, we adopt this probabilistic perspective also for the training of PINNs to analyse the relationship between fitting the data and the physical model. It gives a mean to inform the choice of the loss weighting parameters α_c used in Section 3.5. It will become clear that the concept of PINNs naturally bridges the gap between the often separately treated forward and inverse problems.

We start by defining in Section 4.1 the problem of determining a suitable model $\boldsymbol{f}_\lambda$ and introduce relevant concepts. Section 4.2 then presents the estimation framework that PINNs provide. Within this framework, we first approximate the trajectory in Section 4.3 before estimating the ODE parameters in Section 4.4. We consider alternative approaches in Section 4.5 that underline the attractiveness of PINNs in this context and conclude in Section 4.6.

4.1 Problem setup and concepts

This section briefly introduces the concrete problem setup and the relevant components of the solution approach. As we keep this description very concise, we refer the interested reader to the following textbooks for more detailed descriptions: The author in [102] describes the inverse problem in the context of system identification, in [48] the statistical learning view dominates, and [103] gives an in-depth Bayesian perspective.

4.1.1 Problem formulation

We begin by defining a dataset $\mathcal{D}$ that collects $|\mathcal{D}| = N$ measurements stemming from a dynamical system

$$\mathcal{D} = \{(t^{(1)}, \boldsymbol{y}^{(1)}), (t^{(2)}, \boldsymbol{y}^{(2)}), \dots (t^{(N)}, \boldsymbol{y}^{(N)})\}. \tag{4.1}$$

They map from time $t \in \mathbb{R}$ to the observation $\boldsymbol{y} \in \mathbb{R}^m$. The problem could be extended to include variables other than time as the input, but for a dynamical system these inputs could also be expressed as a time-varying variable in the model $\boldsymbol{f}$. Therefore, the single input dimension of time is sufficient for explaining the observations. We assume that the measurements are generated by

the following dynamical system:

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\lambda}}(t, \boldsymbol{x}) \tag{4.2a}$$

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\varepsilon} \tag{4.2b}$$

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \Sigma). \tag{4.2c}$$

The observations $\boldsymbol{y}$ are linked to the state $\boldsymbol{x} \in \mathbb{R}^n$ by the observation function $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m$, but they are, additionally, subject to normally distributed noise $\boldsymbol{\varepsilon} \in \mathbb{R}^m$ with mean $\boldsymbol{\mu} = \mathbf{0}$ and the covariance matrix Σ.

The assumed model function will be the update function of the classical machine model connected to an infinite bus, i.e., the differential equation used in Chapter 3 and defined in (3.2)

$$\frac{d}{dt}\begin{bmatrix}\delta \\ \Delta\omega\end{bmatrix} = \begin{bmatrix}\omega_0 \Delta\omega \\ \frac{1}{2H}\left(P_m - D\Delta\omega - \frac{EV}{X}\sin\delta\right).\end{bmatrix}$$

Hence, the parameters of this model are the three coefficients $\boldsymbol{\lambda} = [H, D, \frac{EV}{X}]$ which remain to be determined. Furthermore, we assume all states to be directly observable, i.e., $\boldsymbol{g}(\boldsymbol{x}) = I$ with I being the identity matrix. This is chosen for simplicity in the presentation, but is not a required assumption. The covariance matrix Σ is assumed to be unknown but diagonal, that means all states are independently disturbed. Once more, this assumption can be lifted. We will use the notion of the dimensionless state $\tilde{\boldsymbol{x}} = \boldsymbol{T}(\boldsymbol{x})$ also for the observations $\tilde{\boldsymbol{y}} = \boldsymbol{T}(\boldsymbol{y})$ (see Section 3.3). As we do not know the parameters $\boldsymbol{\lambda}$, we cannot use the same transformation $\boldsymbol{T}$ as in the forward problem, however, by instead performing a standardisation $\boldsymbol{T}'$, i.e., subtracting the mean and dividing by the standard deviation, we obtain a comparable effect. When we compute norms in these coordinates, the norm will carry the subscript T, i.e., $\|\cdot\|_T$. Furthermore, we assume that the noise has the same magnitude in each of the dimensionless coordinates, i.e., $\Sigma = \sigma_y I$. A value of $\sigma_y = 0.01$ corresponds to noise of 1% relative to the dimensionless observation $\tilde{\boldsymbol{y}}$.

4.1.2 Maximum likelihood (MLE) and maximum a posteriori (MAPE) estimators

With an inverse problem we aim to obtain a parameter estimate $\hat{\theta}$ from a dataset $\mathcal{D}$. In this and the next section we will use the standard notation of θ to represent a parameter. From Section 4.1.4 onwards, we return to the otherwise used notation, i.e., using $\boldsymbol{\theta}$ for the Neural Network (NN) parameters and $\boldsymbol{\lambda}$ for the ODE parameters. In order to obtain a parameter estimate $\hat{\theta}$, there are two main approaches, namely the Maximum Likelihood Estimator (MLE)

$$\hat{\theta}_{ML} = \arg\max_{\theta} p(\mathcal{D}|\theta) \tag{4.3}$$

and the Maximum a Posteriori Estimator (MAPE)

$$\hat{\theta}_{MAP} = \arg\max_{\theta} p(\theta|\mathcal{D}). \tag{4.4}$$

The MLE originates from a frequentist's perspective where we seek to maximise the likelihood that a certain parametrised model generated the data. The Bayesian approach behind the MAPE, in contrast, assumes that the parameter estimate $\hat{\theta}$ is a distribution itself, and before observing any data, we already have a prior on these model parameters $p(\theta)$. The MAPE selects the mode of this

posterior distribution, but one could also choose, e.g., the expected value of the posterior $\mathbb{E}[p(\theta|\mathcal{D})]$. The estimates obtained from the two viewpoints agree in the limit of many data points, but in settings with scarce data the estimates can differ significantly.

To evaluate the posterior distribution $p(\theta|\mathcal{D})$ for the MAPE, we apply Bayes rule

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{4.5}$$

However, its calculation requires the integration

$$p(\mathcal{D}) = \int_\theta p(\mathcal{D}|\theta)p(\theta)d\theta. \tag{4.6}$$

This integral is analytically only solvable when the two distributions stem from the same family of distributions, known as being conjugate to each other [104]. If this is not possible, we use the unnormalised posterior distribution

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta) \tag{4.7}$$

for which we can obtain an approximation. Section 4.1.3 briefly presents two common approaches. The MAPE then chooses the mode of this approximated distribution, the scaling due to the missing normalisation does not affect the estimate.

When we approximate the posterior, we directly obtain a distribution of estimates from which can quantify uncertainty. In contrast, the MLE only provides a point estimate, but we can calculate a confidence band by utilising the Information Matrix I

$$I(\theta) = -\frac{\partial^2}{\partial\theta\partial\theta^\top} \log p(\mathcal{D}|\theta). \tag{4.8}$$

As it is the Hessian of the log-likelihood, it captures the curvature at the obtained estimate and hence the sensitivity of the log-likelihood with respect to the parameters. For further details we refer to [48].

4.1.3 Approximating the posterior $p(\theta|\mathcal{D})$

The evaluation of the posterior distribution $p(\theta|\mathcal{D})$ poses the great computational challenge in Bayesian statistics. Here, we will only give a very high-level overview as the algorithmic details are out of the scope of this thesis. We refer to [103] for a comprehensive introduction to the topic. Two standard approaches are sampling-based and distributional approximations. For this work, we use a Hamiltonian Monte-Carlo (HMC) sampling-based approach, based on the observations in [46] regarding the approximation quality.

HMC-based algorithms utilise the idea to approximate a distribution by drawing samples, i.e., using a Monte-Carlo approach. As this can be very inefficient for high-dimensional and complex distributions, Markov-Chain Monte-Carlo (MCMC) methods use previous samples to inform where the next sample should be drawn. In HMC algorithms, this relation is described by forming an auxiliary Hamiltonian dynamical system [105]. The specific implementation used in this work is the No-U-turn sampler (NUTS) [106] implemented in NumPyro [107], [108]. The described improved sampling algorithms still face the challenge of tractability. Furthermore, it can be challenging to determine suitable hyperparameters and to decide whether the algorithm converged to a stable approximation of the posterior.

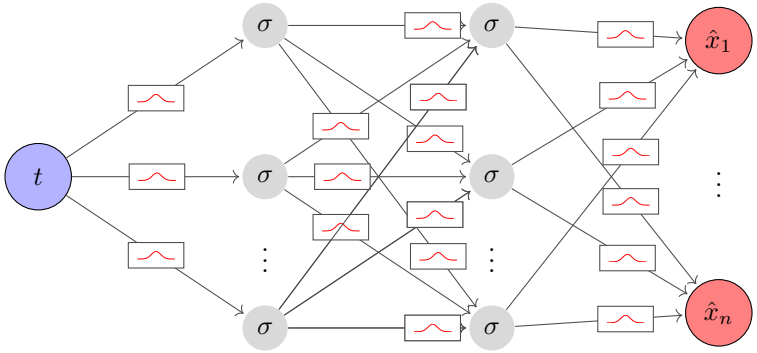


Figure 4.1: Architecture of a Bayesian Neural Network (BNN).

A more lightweight alternative can be variational inference (VI) algorithms, that approximate the posterior with simpler distributions to improve tractability. Algorithms such as stochastic variational inference [109] and Stein variational inference [110] allow the scaling to large datasets. Variational inference algorithms hinge all around the choice of the "simpler" distribution to express calculations analytically, but it can lead to limitations with respect to the degree of complexity of the posterior distribution.

4.1.4 Bayesian Neural Network (BNN)

If we view learning the parameters $\boldsymbol{\theta}$ of a NN, such as in Chapter 3, from a probabilistic viewpoint, we are obtaining a MLE of these parameters. Under the assumption of Gaussian noise, the MLE is obtained by minimising the familiar loss function

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}|} \sum_{i \in |\mathcal{D}|} \left\| \hat{\boldsymbol{x}}^{(i)} - \boldsymbol{x}^{(i)} \right\|_{T,2}^2 \tag{4.9}$$

$$= \arg\min_{\boldsymbol{\theta}} \mathcal{L}_x \tag{4.10}$$

Appendix C provides a short derivation, it can also be found in standard textbooks, e.g., [49]. If we use this MLE without any further modifications, we easily obtain overfitting models that do not generalise well. Hence, we include inductive biases such as regularisation and early stopping to prevent such outcomes and improve generalisation. This can be viewed as a prior in a Bayesian setting as elaborated many times [64], [111], [112]. Thereby, we obtain a MAPE but without approximating the posterior explicitly. This eases the computation, but we loose the additional information that the posterior can provide with respect to uncertainty estimates.

In a BNN [113], in contrast, the weight and bias parameters $\boldsymbol{\theta}$ that define the linear transformations of the layers (see Section 2.3) are considered to be distributions instead of single values. Thereby, we define a distribution of NN models for a specific hypothesis class h_θ. Figure 4.1 provides a schematic illustration, we omitted the bias variables in the layers. By using marginalisation over these distributions of $\boldsymbol{\theta}$, we aim to find suitable weight distributions, i.e., the posterior of the NN parameters. The authors in [114] elaborate on the difficulties that arise in the approximation of posterior when using Markov-Chain Monte-Carlo-based methods and other important aspects such as the choice of a prior.

4.2 The PINN estimation approach

The approach of using PINNs for inverse problems, as introduced in [26], can be split into two flows of information as visualised in Figure 4.2. The first one, represented in the upper half, is the estimation of a trajectory. It is based on the dataset of measurements $\mathcal{D}$, i.e., t and $\boldsymbol{y}$. We use a NN with hypothesis class h_θ and a parameter set $\boldsymbol{\theta}$ to estimate the trajectory $\hat{\boldsymbol{x}}$ for the input values t. After applying the observation function $\boldsymbol{g}(\boldsymbol{x})$, we can form the model likelihood based on $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and σ_y. Using this likelihood, and potentially the priors $p(\boldsymbol{\theta})$ and $p(\sigma_y)$, allows the estimation of the parameters $\boldsymbol{\theta}$ and σ_y. The second flow of information, shown in the lower half, evaluates the continuous trajectory estimate on the collocation points from $\mathcal{D}_c$, i.e., we obtain $\hat{\boldsymbol{x}}_c$ and with Automatic Differentiation (AD) also $\frac{d}{dt}\hat{\boldsymbol{x}}_c$. By calculating the difference $\hat{\boldsymbol{f}} = \boldsymbol{f}_{\boldsymbol{\lambda}}(\hat{\boldsymbol{x}}_c) - \frac{d}{dt}\hat{\boldsymbol{x}}_c$, we can form another likelihood function, that depends on $\boldsymbol{\lambda}$, $\boldsymbol{\theta}$, and the "noise" σ_f for the estimation of the ODE parameters. It allows the identification of the parameters $\boldsymbol{\lambda}$ but it will also affect $\boldsymbol{\theta}$.

This basic setup can be used with or without priors. If they are not included, we use MLEs – the training of NNs and PINNs [26] corresponds to this case. When priors are included, a MAPE is employed; it corresponds to a BNN for the trajectory estimation, i.e., the upper half of the figure, and to a BPINN for the ODE parameter estimation as proposed in [46].

We will first formulate the different components in the workflow in Figure 4.2. Section 4.3 then illustrates the use of NNs and a BNN for the trajectory estimation before Section 4.4 continues with the ODE parameter estimation task, i.e., using PINNs and a BPINN for the estimation of $\boldsymbol{\lambda}$.

The approximation of $\hat{\boldsymbol{y}}$

$$\hat{\boldsymbol{y}} = \boldsymbol{g}(\hat{\boldsymbol{x}}) = \boldsymbol{g}(h_\theta(t)) \tag{4.11}$$

by the hypothesis class $h_\theta = \hat{\boldsymbol{x}}$ parametrised by $\boldsymbol{\theta}$ will be the first objective. The likelihood $p(\mathcal{D}|\boldsymbol{\theta}, \sigma_y)$ under a Gaussian noise assumption with standard deviation σ_y becomes

$$p(\mathcal{D}|\boldsymbol{\theta}, \sigma_y) = \prod_{i=1}^{|\mathcal{D}|} \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{\left\|\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}\right\|_{T,2}^2}{2\sigma_y^2}\right). \tag{4.12}$$

This likelihood is not only dependent on the parameters $\boldsymbol{\theta}$, but also on the estimate of the observation noise (given that it is unknown), hence, we also have to provide an estimate for σ_y. For quantifying the mismatch between observation and prediction, we use the norm in the transformed coordinates. This choice incorporates the idea of measuring the distance in the dimensionless states that we also used in Chapter 3. As we set $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{x}$, we can simply use this interpretation, for a different form of $\boldsymbol{g}(\boldsymbol{x})$ the norm might need to be redefined as well. If we include priors $p(\boldsymbol{\theta})$, $p(\sigma_y)$, they will be a multi-variate normal distribution and a log-normal distribution

$$\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, I) = \frac{1}{\sqrt{(2\pi)^{|\boldsymbol{\theta}|}}} \exp\left(-\frac{1}{2}\left\|\boldsymbol{\theta}\right\|_2^2\right) \tag{4.13}$$

$$\log \sigma_y \sim \mathcal{N}(\sigma_{y,\mu}, \sigma_{y,\sigma}). \tag{4.14}$$

Whenever not explicitly stated, we use $\sigma_{y,\mu} = \log(0.1)$ and $\sigma_{y,\sigma} = 0.25$.

The likelihood function for the ODE model formulates as

$$p(\mathcal{D}_c|\boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma_f) = \prod_{i=1}^{|\mathcal{D}_c|} \frac{1}{\sqrt{2\pi\sigma_f^2}} \exp\left(-\frac{\left\|\hat{\boldsymbol{f}} - \mathbf{0}\right\|_{T,2}^2}{2\sigma_f^2}\right), \tag{4.15}$$

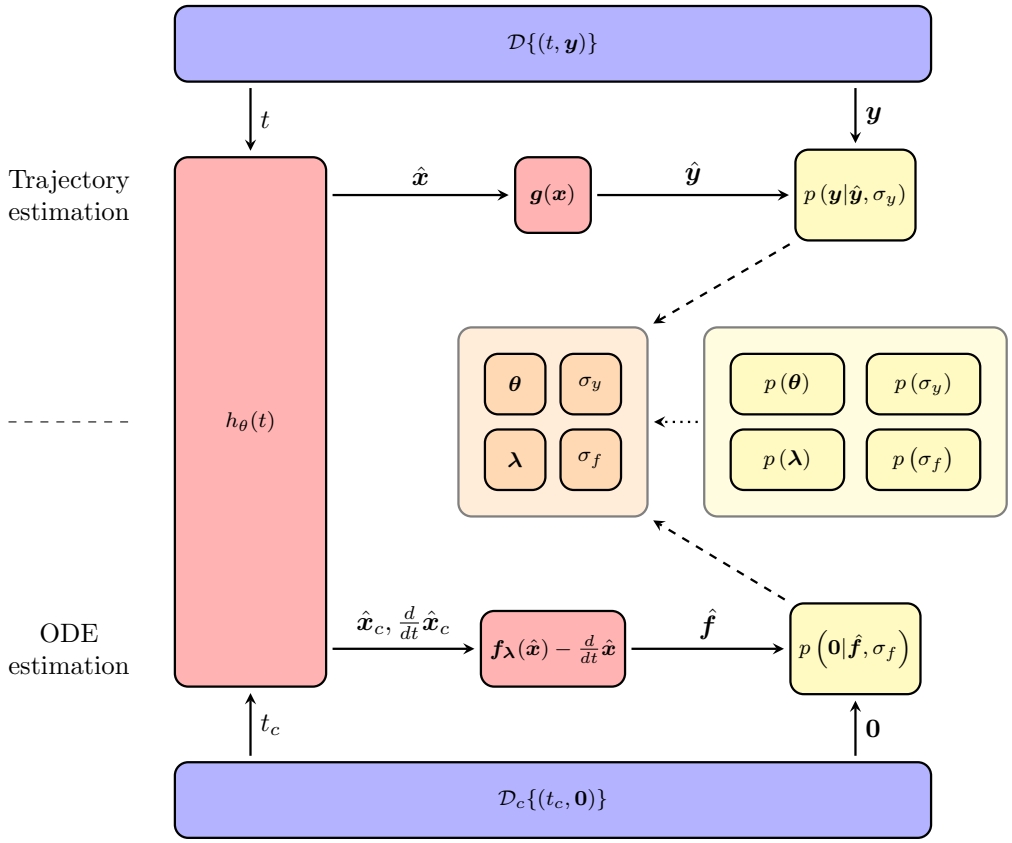


Figure 4.2: Flowchart of the estimation process with PINNs and BPINNs. The blue boxes represent the datasets used, the red boxes stand for deterministic functions. The yellow boxes signify likelihoods or priors and the orange boxes correspond to the model parameters. The upper half of the figure show the quantities and functions involved in the trajectory estimation. The lower half illustrates the process of the ODE parameter estimation.

where $\hat{\boldsymbol{f}} = \boldsymbol{f}_{\boldsymbol{\lambda}}(\hat{\boldsymbol{x}}) - \frac{d}{dt}\hat{\boldsymbol{x}}$ evaluates the mismatch of the left-hand and right-hand side of the ODE, it is the same concept as in Section 3.5.1. Once more, we use the dimensionless states for the calculation of the norm. The likelihood depends on the ODE parameters $\boldsymbol{\lambda}$, the NN parameters $\boldsymbol{\theta}$ (as they determine $\hat{\boldsymbol{x}}$), and an uncertainty estimate on the physics loss σ_f. We will come back to the significance of σ_f in Section 4.4.2. For each of the ODE parameters, we use a Gamma-distribution Γ due its support on the positive numbers and as it has a weakly informative character for our problem

$$\boldsymbol{\lambda}_i \sim \Gamma(3, 1). \tag{4.16}$$

4.3 Estimating a continuous trajectory with NNs and BNNs

We begin by estimating the trajectory which will form the basis for the following estimation of $\boldsymbol{\lambda}$. The test consists of the estimation of the trajectory based on a small and then a larger dataset. We first test a collection of 50 NNs, i.e., 50 vanilla NNs that are trained entirely independent of each other. They all constitute MLEs, but their estimated parameters will differ due to the different initialisation of the parameters $\boldsymbol{\theta}$. The initialisation is based on a Glorot-Normal distribution [76]. On the other hand, we train a BNN on the same dataset. We choose this comparison to highlight

that the collection of NNs and samples from the BNNs are not equivalent in the estimation that they provide[1].

Collection of NNs

The first dataset has size $|\mathcal{D}| = 9$. Figure 4.3 illustrates the range of the predictions between the 5th and 95th percentile (blue shade) and the median prediction (blue line) evaluated at different epochs represented in the three columns. The percentiles are evaluated on the predictions, independently

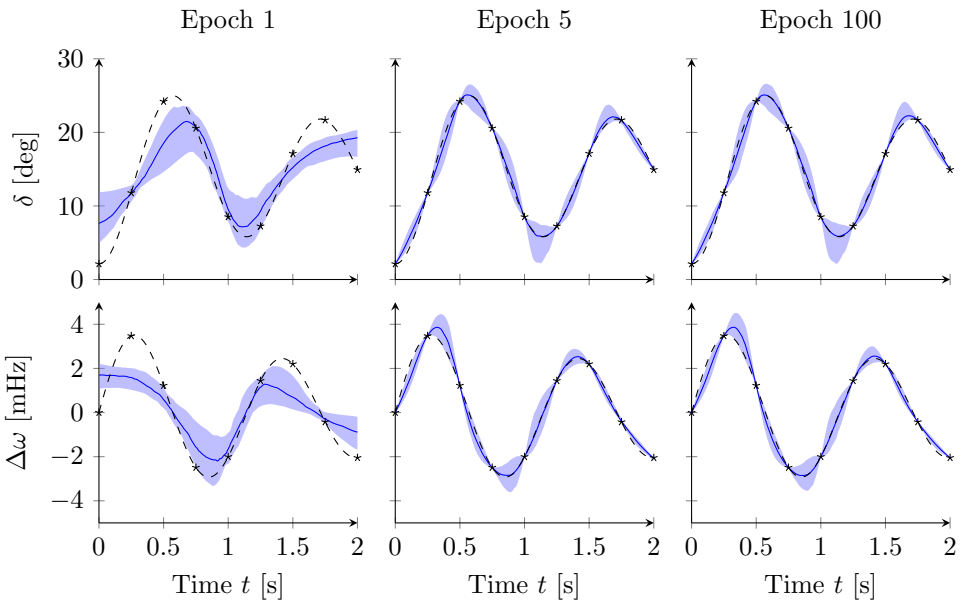


Figure 4.3: Trajectory estimation based on a collection of 50 NNs trained on a noise-free dataset with $|\mathcal{D}| = 9$. The columns show the epoch, i.e., the training progress.

for each point in time. This means that the median line does neither correspond to a "median parameter set" nor a single model; we calculate statistics over the collection of predictions. The true trajectory is shown as the dashed black line and the stars indicate the supplied data points. After one epoch the different NNs already capture the main trend in the data but still show some variation. After a few more epochs all points are exactly fit and, across the different models, we only observe small variations in between the data points. Training longer does not have an effect here, as we reached the optimiser's tolerance. In contrast, for a larger ($|\mathcal{D}| = 41$) and noisy ($\sigma_y = 0.2$) dataset as shown in Figure 4.4, the models visibly overfit after 100 epochs. At an earlier epoch (5), the models are in great agreement and yield a much better fit, knowing the target trajectory. Given the present noise, the loss value at this epoch would be acceptable, but without a validation dataset or a fixed early stopping criterion, the optimisation proceeds to also fit the noise.

[1] Using a collection of models, here NNs, to improve predictions is a concept that forms the basis of many powerful learning approaches. Such methods, that are based on, e.g., bagging, bootstrapping, boosting, ensembles, generate various candidate models that are then combined in a specific way to obtain a prediction and often even an uncertainty quantification. If there is the right amount of diversity in the models, such approaches implicitly yield an approximation of posterior distribution in the Bayesian setting. We refer to [48] for an overview of the methods and an in-depth treatment of the associated Bayesian interpretation. In the presented case, the varied initialisation of the NN parameters does not lead to enough diversity in the models. Therefore, an uncertainty quantification based on the collection of NNs is not informative.

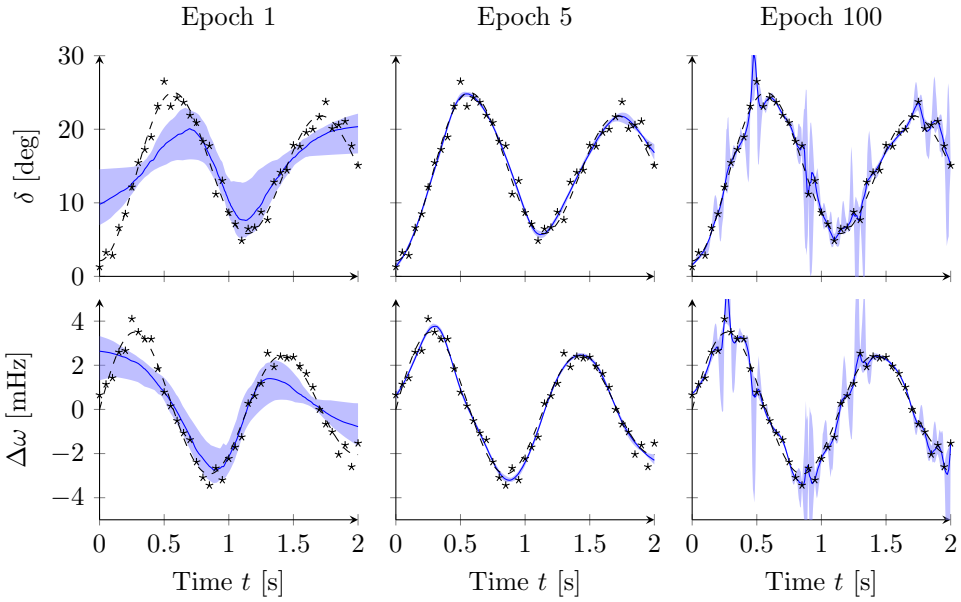


Figure 4.4: Trajectory estimation based on a collection of 50 NNs trained on a noisy dataset with $|\mathcal{D}| = 41$. The columns show the epoch, i.e., the training progress.

BNNs

We repeat the experiment with exactly the same small and larger dataset with a BNN, shown in Figures 4.5 and 4.6. The coloured area represents again the range between the 5th and 95th

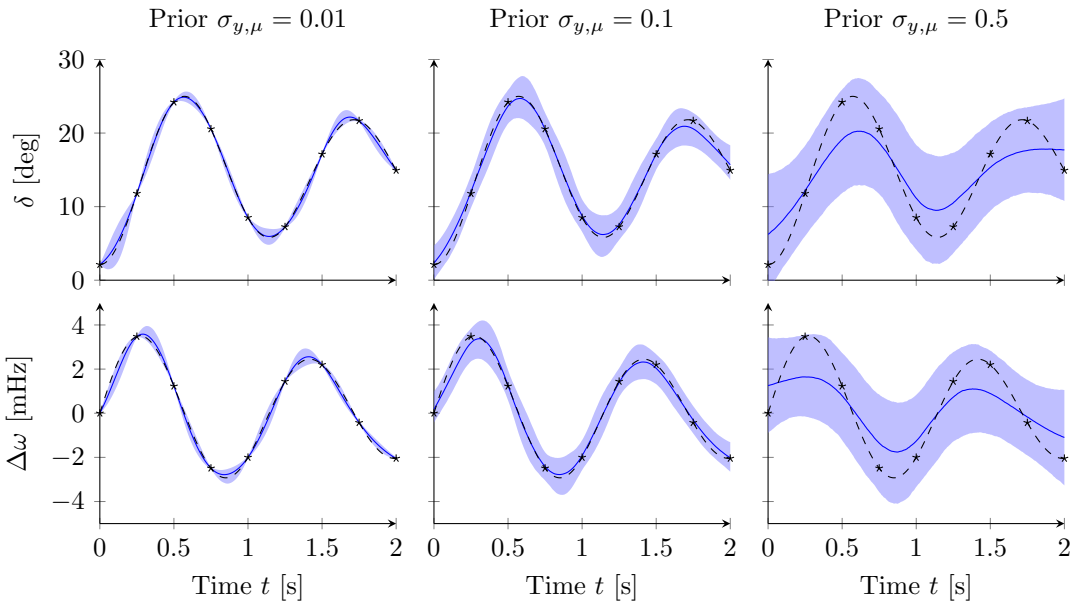


Figure 4.5: Trajectory estimation using a BNN trained on a noise-free dataset with $|\mathcal{D}| = 9$. The columns show different prior settings.

percentile of the estimated trajectories[2]. These predictions now stem from the "sampled models"

[2]This corresponds to the posterior distribution, not the posterior predictive distribution as we do not marginalise over the noise estimate for the plot. This would be relevant for predicting the next observed value, we instead care about the noise-free estimate which we will use for the inference of the ODE parameters $\boldsymbol{\lambda}$.

that we collected when applying the HMC algorithm for the estimation of the posterior distribution[3]. The number of drawn samples should be chosen such that the resulting statistical moments of the posterior do not change. For this reason, the HMC algorithm has a so-called warm-up phase. The samples generated during this time are not used for the trajectory estimation. The three different columns now correspond to different parametrisations $\sigma_{y,\mu}$ of the prior $p(\sigma_y)$ on the observation noise.

We can clearly observe a strong effect of the prior for the small dataset. For a tight prior (small $\sigma_{y,\mu}$), the data points are fit nearly exactly and an uncertainty is only visible in between the points. The widest prior, in contrast, leads to a significant bandwidth around the observations. Even at the point itself, the prior causes the model to not fit the data points exactly. As we gather

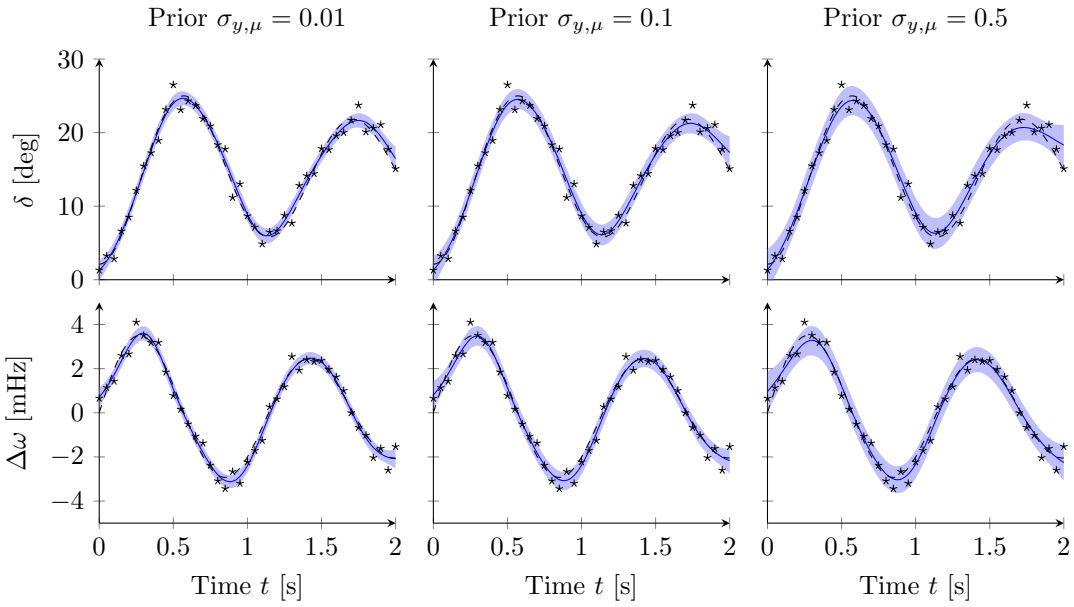


Figure 4.6: Trajectory estimation using a BNN trained on a noisy dataset with $|\mathcal{D}| = 41$. The columns show different prior settings.

more evidence, e.g., from a larger dataset, we gain more independence from the prior as shown in Figure 4.6. This illustrates the importance of a prior well, as we can only overturn it when there is enough evidence present, as it is the case for the larger dataset. With few data points there is simply no way to decide upon the magnitude of the noise, hence, we rely on the prior. But even for large datasets the choice of an appropriate prior is important due to its effect on the estimation algorithm in terms of accuracy and speed of convergence to a stable posterior.

Noise and loss estimation

These observations are best explained by returning to the formulation of the MLE and MAPE. By applying the natural logarithm to the MLE in (4.3), we obtain the following optimisation problem

[3]We, of course, sample the parameters $\boldsymbol{\theta}$ and only thereby a model. However, the parameters are not interchangeable across models, therefore, "sampling models" provides the better idea.

(see Appendix C for the derivation) for the parameters $\boldsymbol{\theta}$ of the NN

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)} \right\|_{T,2}^2 \tag{4.17}$$

$$= \arg\min_{\boldsymbol{\theta}} \mathcal{L}_y. \tag{4.18}$$

The optimisation corresponds to the usual training encountered in Chapter 3 (we there use $\mathcal{L}_x$ to denote this loss as it is based on the state $\boldsymbol{x}$, but the formulation otherwise matches). The noise estimate $\hat{\sigma}_y$, however, does not occur in this formulation, but it is directly dependent on the loss value $\mathcal{L}_y$

$$\hat{\sigma}_y = \arg\min_{\sigma_y} \frac{1}{2} \log\left(2\pi\sigma_y^2\right) + \frac{1}{2\sigma_y^2} \left(\min_{\boldsymbol{\theta}} \mathcal{L}_y \right) \tag{4.19}$$

$$= \sqrt{\mathcal{L}_y(\hat{\boldsymbol{\theta}})}. \tag{4.20}$$

This relation explains why we need a validation dataset: A NN with sufficient representation capacity could fit nearly all points perfectly, i.e., $\mathcal{L}_y$ would approach 0, and in that case the noise estimate $\hat{\sigma}_y$ would also be nearly 0. By evaluating a separate validation dataset, we effectively obtain a noise estimate independent of the training dataset, and can then stop the training early as soon as $\hat{\sigma}_y$ increases, i.e., the validation loss increases. We can also turn the argument around: If the noise estimate is accurate, we know the value the loss should approximately assume for the model to not overfit.

If we instead run a MAPE by adding a prior on the noise and the NN parameters, we obtain the following optimisation

$$\begin{aligned} \hat{\boldsymbol{\theta}}, \hat{\sigma}_y = \arg\min_{\boldsymbol{\theta}, \sigma_y} & \log\left(2\pi\sigma_y^2\right) + \frac{1}{|\mathcal{D}|} |\boldsymbol{\theta}| \log(2\pi) + \frac{2}{|\mathcal{D}|} \log(\sigma_y) \\ & + \frac{1}{|\mathcal{D}|} \log(2\pi\sigma_\sigma^2) + \frac{1}{\sigma_\sigma^2} \frac{1}{|\mathcal{D}|} (\log(\sigma_y - \mu_\sigma))^2 \\ & + \frac{1}{{\sigma_y}^2} \left(\frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)} \right\|_{T,2}^2 + \frac{{\sigma_y}^2}{|\mathcal{D}|} \|\boldsymbol{\theta}\|_2^2 \right) \end{aligned} \tag{4.21}$$

The term in the last row in the above formula corresponds to a L^2-regularisation of the NN parameters $\boldsymbol{\theta}$. Its influence on the solution decreases as the dataset size increases, hence, its influence will be strongest in small dataset settings. If we fixed σ_y, we could optimise the bracket over $\boldsymbol{\theta}$, i.e., perform a L^2-regularised NN training, without the complicating noise estimate σ_y. When choosing a regularisation strength as a hyperparameter, we apply exactly this trick – we implicitly assume and fix a noise estimate σ_y. As soon as σ_y is also free to choose, the problem becomes more intricate. These considerations are well established, see e.g. [111], but they will help us in the understanding of the following section on estimating the ODE parameters $\boldsymbol{\lambda}$.

4.4 PINNs and BPINNs for estimating ODE parameters

We now move to the original goal, i.e., estimating the parameters $\boldsymbol{\lambda}$ of the ODE. We return to the likelihood of the ODE model (4.15). However, since it also depends on the NN parameters $\boldsymbol{\theta}$ we have to include the likelihood of the trajectory estimation (4.12), which leads to the joint likelihood

$$p(\mathcal{D}, \mathcal{D}_c | \boldsymbol{\theta}, \sigma_y, \boldsymbol{\lambda}, \sigma_f) = p(\mathcal{D} | \boldsymbol{\theta}, \sigma_y)\ p(\mathcal{D}_c | \boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma_f). \tag{4.22}$$

Corresponding to the previous section, we can again choose to pursue an approach based on a MLE or a MAPE. For the former the NNs are turned into PINNs and for the latter the BNN is turned into a BPINN. We first discuss the resulting parameter estimates and then show the mathematical reasons for the results.

4.4.1 Accuracy of ODE parameter estimates

We conduct this estimation on the two previously used dataset sizes ($|\mathcal{D}| = 9$ and $|\mathcal{D}| = 41$) but now in a noise-free ($\sigma_y = 0.0$) and noisy setup ($\sigma_y = 0.2$) for each of them. We begin by explaining the results shown in Figure 4.7 that were obtained on the noisy small dataset. The x-axis plots the relative estimation error, i.e., we aim for a value of 0. As discussed earlier, a PINN corresponds

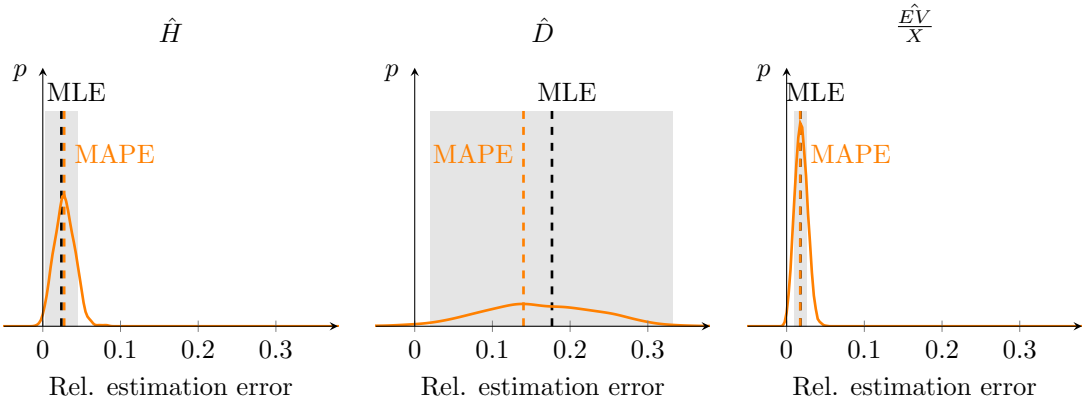


Figure 4.7: Comparison of the relative estimation error $(\hat{\lambda}_i - \lambda_i)/\lambda_i$ for a single PINN (black MLE and confidence interval in grey) and a BPINN (posterior distribution and MAPE in orange) based on a noisy dataset ($|\mathcal{D}| = 9$, $\sigma_y = 0.2$). The panels represent the different parameters.

to a MLE, shown in Figure 4.7 as the black dashed line. It is a point-estimate but by using the Information matrix I for the obtained estimate, we can provide a 95% confidence interval[4] around this estimate, indicated by the grey shaded area. By design, we obtain a posterior distribution for a BPINN, shown as the orange solid line. We derive the MAPE by determining the mode of the distribution, indicated by the orange dashed line.

The MLE and MAPE of a PINN and the BPINN match fairly well and so does the confidence interval with the range of the posterior. We now compare (as in the previous section) a collection of 50 PINNs against the BPINN estimate. In the following, the boxplots represent the distribution of the MLE of the 50 PINNs, we do not plot their confidence intervals to avoid clutter. For the BPINN, the boxplot displays the same distribution as seen in Figure 4.7. The resulting parameter estimates are shown in the four panels in Figure 4.8. The panels along the vertical axis have the same dataset size and the panel along the horizontal axis share the noise scale. The grouping of the three boxplots corresponds to the estimates for the ODE parameters $\boldsymbol{\lambda} = [H, D, \frac{EV}{X}]$ (in that order) evaluated as the relative estimation error, i.e., $\frac{\hat{\lambda}_i - \lambda_i}{\lambda_i}$. Note, that the y-axis has different scales for the noise-free and noisy case. For the noise-free case the estimates are highly accurate for the collection of PINNs and the BPINN. For the noisy case on the small dataset, we observe

[4]Assuming the percentiles of a standard normal distribution, the range is calculated by $\hat{\boldsymbol{\lambda}} \pm 1.96\sqrt{I(\hat{\boldsymbol{\lambda}})^{-1}_{jj}}$, where jj corresponds indicates the entry of the respective parameter (see (4.8) and [48, p.266])

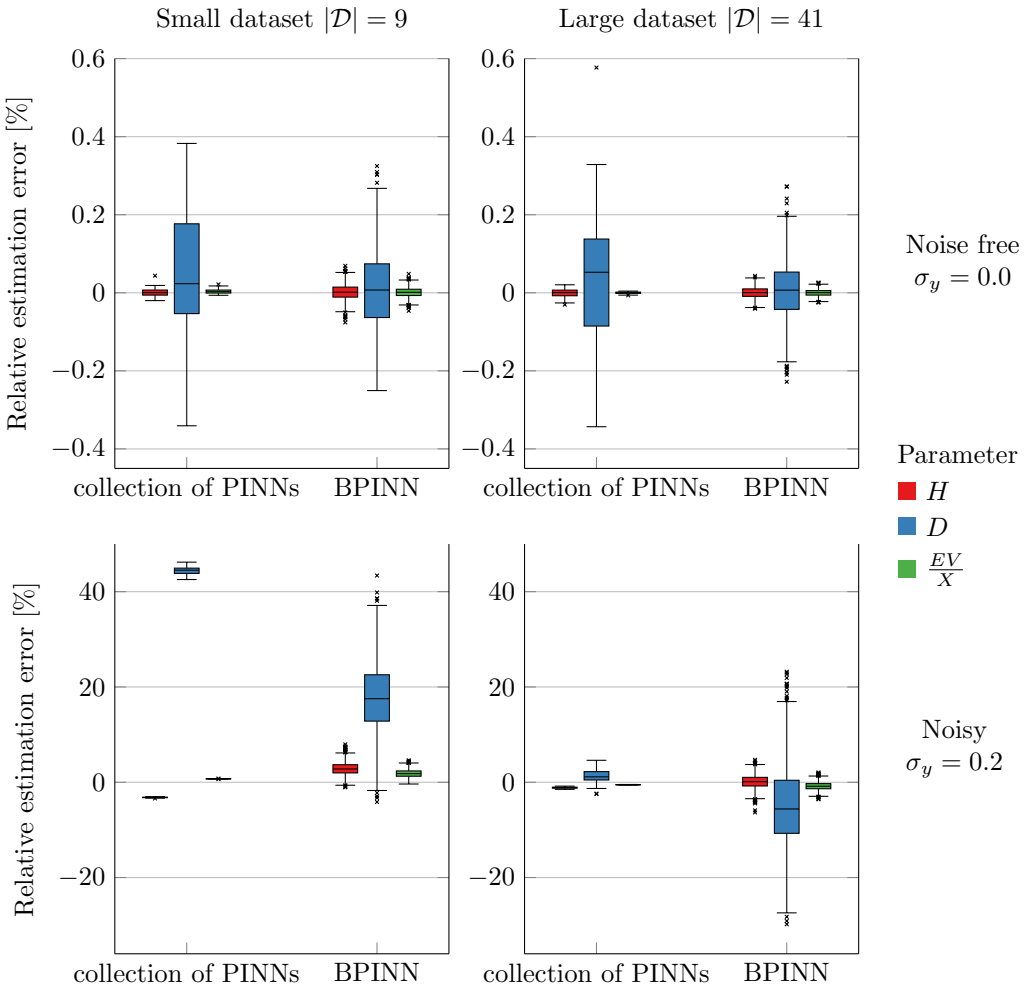


Figure 4.8: Relative estimation error of the ODE parameters $\boldsymbol{\lambda}$ (in the order $[H, D, \frac{EV}{X}]$ in each group of boxplots) for the collection of PINNs and a BPINN. The panels in a column correspond to the same dataset size and in a row to the same noise magnitude.

biased parameter estimates, most notably for the damping coefficient D (Figure 4.7 corresponds to this result for the BPINNs and one of the PINNs). Increasing the dataset size reduces this bias significantly. This improvement can be attributed to a more accurate estimation of the trajectory. For the noise-free case we obtain a near perfect estimation of the trajectory, hence, the parameter estimates can be fitted very accurately. This underlines the importance of the estimation of the trajectory that we discussed in the previous section.

Furthermore, we observe in Figure 4.8 that $\frac{\hat{EV}}{X}$ consistently has the least bias and the lowest variance and D the most. This can be explained by considering their role in the ODE. The coupling between the two states δ and $\Delta\omega$ acts through $\frac{EV}{X}$ (we assume ω_s to be known). Therefore, data points from both states provide evidence. The parameters H and D, in contrast, are entirely dependent on an accurate approximation of the trajectory of $\Delta\omega$. If it is given (as for the noise-free cases), we obtain a very accurate estimate; if not, it leads to errors in the estimates. Here, understanding the trajectory as a damped oscillation helps. The frequency of this oscillation is mostly determined by H and the damping by D. When going back to Figure 4.5, the visual impression suggests that the frequency of the oscillation can be easier identified than the damping of the trajectory. Therefore,

the estimation of H is more accurate than the one of D.

What may surprise is how close the MLE and MAPE are and how well the confidence interval matches the posterior distribution. Furthermore, the MLEs are very consistent as we saw in Figure 4.8 with respect to the initialisation of the PINN. The reason stems from the fact that the dataset of collocation points, we chose $|\mathcal{D}_c| = 201$, provides a lot of evidence. Under such "evidence-rich" circumstances the frequentist's and Bayesian perspectives align and should lead to the same outcome as our observations confirm. Still, the dataset size $|\mathcal{D}|$ plays an important role. For the noisy datasets we could only reduce the bias in the parameter estimates when the dataset size grew as the estimated trajectory only then coincided with the true one. In this regard, the BPINN has a slight advantage due to the implicit estimation of the noise σ_y. If we fixed this value (or set a very tight prior), we can enforce variation in the trajectory estimation and hence we might encounter the "true" trajectory and its parameters.

4.4.2 Balancing the weighting hyperparameter α

Many of the above observations can be explained by taking a closer look at the formulations of the MLE for which we want to find the parameters that maximise (4.22)

$$\hat{\boldsymbol{\theta}}, \hat{\sigma}_y, \hat{\boldsymbol{\lambda}}, \hat{\sigma}_f = \arg\max_{\boldsymbol{\theta}, \sigma_y, \boldsymbol{\lambda}, \sigma_f} p(\mathcal{D}, \mathcal{D}_c | \boldsymbol{\theta}, \sigma_y, \boldsymbol{\lambda}, \sigma_f). \tag{4.23}$$

We can turn this again into the familiar form (derivation in Appendix C)

$$\hat{\boldsymbol{\theta}}, \hat{\sigma}_y, \hat{\boldsymbol{\lambda}}, \hat{\sigma}_f = \arg\min_{\boldsymbol{\theta}, \sigma_y, \boldsymbol{\lambda}, \sigma_f} \log\left(2\pi\sigma_y^2\right) + \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \log\left(2\pi\sigma_f^2\right) + \frac{1}{\sigma_y^2}\left(\mathcal{L}_y + \frac{\sigma_y^2}{\sigma_f^2}\frac{|\mathcal{D}_c|}{|\mathcal{D}|}\mathcal{L}_c\right). \tag{4.24}$$

by setting $\frac{\sigma_y^2}{\sigma_f^2}\frac{|\mathcal{D}_c|}{|\mathcal{D}|} = \alpha_c$

$$\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\lambda}} = \arg\min_{\boldsymbol{\theta}, \boldsymbol{\lambda}} \mathcal{L}_y + \alpha_c \mathcal{L}_c. \tag{4.25}$$

This is the same optimisation problem as in (3.16) for the forward problem of Chapter 3 apart from the fact, that we there know the parameters $\boldsymbol{\lambda}$ (and without the dt-loss). In (4.25), we fixed the estimates for σ_y and σ_f and instead use the tunable hyperparameter α_c. It is the same idea as for the L^2-regularisation in Section 4.3. However, this time, we can make a very insightful argument for the choice of α_c.

For every parameter set $\boldsymbol{\lambda}$, there is a unique trajectory given an initial condition (see Section 2.2). Whenever there is a mismatch in the joint estimation of the trajectory and ODE parameters $\boldsymbol{\lambda}$, i.e., $\mathcal{L}_c > 0$, we should adjust the estimate for $\boldsymbol{\lambda}$ and/or $\boldsymbol{\theta}$ in order to reduce $\mathcal{L}_c$. If the dataset is noisy, we will eventually need to increase $\mathcal{L}_y$ to drive $\mathcal{L}_c$ to 0. The estimated trajectory will then be (nearly) in line with the solution to the ODE parametrised by $\boldsymbol{\lambda}$, i.e., $\boldsymbol{f}_{\boldsymbol{\lambda}}$. We achieve this by increasing α_c, i.e., placing more weight on the physics loss $\mathcal{L}_c$. If we assumed, all parameters in α_c except for σ_y to be fixed, an increase in α_c implies an increase in σ_y. This will eventually lead to $\boldsymbol{y}$ defaulting to the mean of the dataset; the effect of increasing the prior in Figure 4.5 hints at this. We basically "ignore" the dataset and fit a steady state solution. On the other hand, the estimated parameters $\boldsymbol{\lambda}$ and the estimated trajectory $\hat{\boldsymbol{x}}$ either match or not. It is a binary choice, which in probability terms corresponds to a delta Dirac-distribution δ, a unit impulse. This requires that σ_f in (4.15) needs to approach 0 to obtain such a Dirac-distribution. As a consequence, α_c would

grow when keeping the other elements of α_c constant. This puts us in the dilemma, that enforcing the ODE can lead to ignoring the dataset.

For this reason, we employed the fade-in in Chapter 3: We first aim to find a plausible trajectory based on the data, and then start to put more emphasis on obeying the governing ODE. For the noise free case the two objectives align, therefore, the learning outcome is less sensitive to the choice of α_c, except for too high values that can lead to the equilibrium solution. For noisy datasets, α_c determines how easily we can find the correct noise estimate σ_y. For a low value of α_c, we easily obtain a trajectory that fits the data points but it does not correspond to an ODE of the pre-defined structure. As soon as we enforce the physics more, we can obtain a consistent estimation of the parameters and the dataset, but for too high values of α_c, the optimisation algorithm might never come close to that region and hence does not find this optimum.

It is also important to note, that for the noisy case, we are not guaranteed to find the true ODE parameters, even for large values of α_c. We only get a parameter set that would be consistent with the estimated trajectory. If this trajectory estimate is inaccurate, so will be the parameter estimate. The posterior should assign the true values a non-zero probability, but we would not choose it as the MAPE.

The authors in [79] show how α_c impacts the learning of forward problems. Their analysis focuses on the size of the gradients of the different loss terms with respect to the parameters and leads to an adaptive weighting scheme for α_c. In [115], an adaptive weighting scheme is proposed for BPINNs based on analysing the training under the lens of multitask learning.

4.5 Related approaches

The task of system identification or parameter estimation has a long-standing history in the context of dynamical systems. For a comprehensive introduction we refer to [102] and [51] which summarises recent important data-driven techniques. Nonetheless, we want to briefly describe two approaches a little more in detail – Sparse Identification of Non-Linear Dynamics (SINDy) and Neural Ordinary Differential Equation (NeuralODE) – as they represent relevant approaches in the recent developments. This description serves the purpose of clarifying methodological differences of PINNs and BPINNs with respect these other methods. The main line of distinction can be drawn along the question, how much structure and of which form we impose on the ODEs, i.e., on the function $\boldsymbol{f}_{\boldsymbol{\lambda}}$. The above presented PINNs and BPINNs require the entire structure to be known, SINDy provides an approach that allows to identify a structure out of a larger library of potential contributing terms, and NeuralODEs can be entirely structure-free. By the choice of method, we select how much structure we impose on the ODEs. In turn, this will determine the amount of data that we need for useful estimates. Striking this balance between model capacity and generalisation mirrors the discussions in Section 2.1.

4.5.1 Sparse Identification of Non-linear Dynamics (SINDy)

The idea behind SINDy [116] is that we first compute a *library* of basis functions $\Theta(\boldsymbol{x})$ that can be linear and non-linear functions, e.g., trigonometric or polynomial combinations, of the states $\boldsymbol{x}$. Based on this library, we perform a linear regression to find a sparse set of parameters $\boldsymbol{\xi}$ that describes the change of the state vector $\frac{d}{dt}\boldsymbol{x}$ as a linear combination of the library terms

$$\frac{d}{dt}\boldsymbol{x} = \Theta(\boldsymbol{x})\boldsymbol{\xi}. \tag{4.26}$$

Based on this setup, a linear regression is performed that is regularised to promote sparsity in $\boldsymbol{\xi}$

$$\min_{\boldsymbol{\xi}} \left\| \frac{d}{dt}\boldsymbol{x} - \Theta(\boldsymbol{x})\boldsymbol{\xi} \right\|_2 + \lambda \left|\boldsymbol{\xi}\right| \tag{4.27}$$

In the above, a regularisation on the parameter size weighted with λ is used. Overall this setup provides a very scalable and light-weight parameter estimation technique. Compared to PINNs, the calculation or approximation of the derivative $\frac{d}{dt}\boldsymbol{x}$ in settings with few or very noisy data is a challenge. The continuous-in-time nature can free PINN estimates of this difficulty. Needless to say, that advanced versions of SINDy are developed to deal with such settings. However, the approximation needs to rely on numerical differentiation, whereas as PINNs can utilise AD. The other challenge arises from the selection of the library terms and the desired level of sparsity. To incorporates a Bayesian perspective, the authors in [117] use an ensemble strategy. The authors also provide an good overview of recent applications and developments of SINDy, that we refer the interested reader to. For the case presented in the previous sections, we could also apply SINDy. Since we know the library terms $\Theta(\boldsymbol{x})$ in advance, the regularisation would not be necessary. The challenge would arise from the noise and the few data points as they complicated the calculation of $\frac{d}{dt}\boldsymbol{x}$. In turn, PINNs require the full structure of the ODE to be known. Allowing for a variety of possible structures, as SINDy does, could serve as inspiration to develop PINNs further.

4.5.2 NeuralODEs

NeuralODEs provide a very generic and flexible approach to learning update functions. They were introduced in [100] and [118] provides an excellent deep dive into the topic with various extensions. In a NeuralODE, we train the update function $\boldsymbol{f}$ as a black box, a hypothesis class h_θ that we can design freely, instead of learning particular parameters $\boldsymbol{\lambda}$ to fit a given functional form.

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\lambda}}(t, \boldsymbol{x}) = h_\theta(t, \boldsymbol{x}) \tag{4.28}$$

NeuralODEs are most interesting when there is no clear and appropriate functional form available (still in the context of ODEs), a classic setup of Reduced-Order Modelling (ROM). However, being very flexible (or unstructured) comes at the price of requiring a high number of data points for the training as [118] reports. The hypothesis class h_θ needs to learn the entire vectorfield that is otherwise described by a few parameters in a set of ODEs. Hybrid approaches that use NeuralODEs only for fitting "unexplainable" residuals in a dynamical system emerged as an alternative to combine very well generalising ODEs with the expressive power of NeuralODEs [119]. After training, a NeuralODE can be seamlessly included in any ODE solver algorithm, e.g., a Runge-Kutta (RK) scheme, as they are based on the evaluation of the update function $\boldsymbol{f}$ without requiring knowledge of the underlying structure. As both PINNs and NeuralODEs use NNs and concern ODEs, there can be confusion about what task is solved. By referring back to the illustration in Figure 3.25, we can clarify the distinction. NeuralODE are approximations of the update function $\boldsymbol{f}$ and then need a separate ODE solver to evaluate a trajectory. PINNs approximate the entire trajectory and can then also fit a parametrised update function $\boldsymbol{f}_{\boldsymbol{\lambda}}$ – the two approaches constitute two distinct "poles" of solution techniques.

4.6 Concluding remarks

We have shown in this chapter that PINNs can naturally cover the inverse problem similarly well in a point-estimate setting and for estimating a posterior distribution. The high quality of the

estimates and the robustness with respect to noise can be attributed to the approach of PINNs to directly express the trajectory as an explicit function. This avoids discretisation or alleviates noise corruption issues to a large extent. Furthermore, the probabilistic view provides us with rich insights into the interpretation of learning dynamical systems, the impact of the parameters $\boldsymbol{\lambda}$, and the weighting of the different loss terms, in particular the weighting of the physics loss $\mathcal{L}_c$ with α_c.

Future work can, on one hand, explore developing the methodology further. The integration of low- and high-fidelity information, irregular sampling, and partial observations are natural extensions of the presented results. Furthermore, the exploration of the full Bayesian modelling in settings with very few observations is interesting. For more data-rich setting, it remains to be seen if PINNs as MLEs can reliably match the MAPEs based on BPINNs. On a more theoretical level, the probabilistic view of learning might be valuable to investigate phenomena observed in the forward problem, similar to the loss weight α_c. In particular, an analysis of the loss landscape under the lens of Bayesian inference could be fruitful. Insights could inform decisions on the optimiser, but also on the loss function, the dataset, and the hypothesis class, i.e., help designing powerful inductive biases for the forward problem.

Part II

PINNs for power system computations

CHAPTER 5

Multi-machine time domain simulations

We focus in the following chapters of this thesis on the intended use case of Physics-Informed Neural Networks (PINNs), i.e., the application to PINNs for Time-Domain Simulations (TDSs) to power system dynamics. This chapter introduces the relevant machine models in Section 5.1. Section 5.2 describes how they are combined in a multi-machine power system model which forms a system of Differential Algebraic Equations (DAEs). Section 5.3 presents relevant solution approaches, in particular so-called Semi-Analytical Solution (SAS) methods, which we build on in Chapter 7. Lastly, Section 5.4 briefly introduces the test cases used in the two subsequent chapters.

5.1 Component modelling

The component model, that we investigate, is the "classical" machine model which is a simplification of the two-axis generator model. It constitutes a well-studied model for synchronous generators, and it is derived from much more detailed descriptions of the electro-magnetic and electro-mechanic phenomena. Sauer and Pai provide in [4] a comprehensive overview of this modelling and we for the most part follow their notation.

Two-axis model

The update equation for the two-axis model formulates as

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}\left(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{y}; \boldsymbol{\lambda}\right) \tag{5.1}$$

$$\frac{d}{dt}\begin{bmatrix} E'_q \\ E'_d \\ \delta \\ \Delta\omega \end{bmatrix} = \begin{bmatrix} \frac{1}{T'_{do}}\left(-E'_q - (X_d - X'_d)I_d + E_{fd}\right) \\ \frac{1}{T'_{qo}}\left(-E'_d + (X_q - X'_q)I_q\right) \\ \omega_s \Delta\omega \\ \frac{1}{2H}\left(P_m - E'_d I_d - E'_q I_q - (X'_q - X'_d)I_d I_q - D\Delta\omega\right) \end{bmatrix} \tag{5.2}$$

The differential state $\boldsymbol{x} = \begin{bmatrix} E'_q & E'_d & \delta & \Delta\omega \end{bmatrix}^\top$ includes the voltages E'_q and E'_d along the d-q-axes, the rotor angle δ and the frequency deviation $\Delta\omega = \omega - \omega_s$ from the system frequency ω_s. The exciter voltage E_{fd} and the mechanical power P_m from the turbine shaft constitute the control input $\boldsymbol{u} = \begin{bmatrix} E_{fd} & P_m \end{bmatrix}^\top$. The time constants T'_{do}, T'_{qo}, the reactances X_d, X'_d, X'_q, the machine inertia H, and damping constant D form the machine parameters $\boldsymbol{\lambda} = \begin{bmatrix} T'_{do} & T'_{qo} & X_d & X'_d & X'_q & H & D \end{bmatrix}^\top$. We formally include time t in the update function $\boldsymbol{f}$, but we assume an autonomous system, i.e., no dependence on t. The stator currents (I_d and I_q) in the machine reference frame, which is denoted by lower case subscripts d and q, are calculated as

$$\begin{bmatrix} I_d \\ I_q \end{bmatrix} = \begin{bmatrix} R_s & -X'_q \\ X'_d & R_s \end{bmatrix}^{-1} \begin{bmatrix} E'_d - V\sin(\delta - \theta) \\ E'_q - V\cos(\delta - \theta) \end{bmatrix} \tag{5.3}$$

These stator currents are linked to the terminal voltage magnitude V and angle θ through the algebraic equations above. We collect the algebraic variables in $\boldsymbol{y} = \begin{bmatrix}\theta & V\end{bmatrix}^\top$. Later on, we will require the complex current injections $\bar{i}$ to the network in the network reference frame, denoted by capital subscripts D and Q, and calculated as

$$\bar{i} = (i_D + j\, i_Q) = (I_d + j\, I_q) e^{j(\delta - \pi/2)}. \tag{5.4}$$

Classical machine model

The two axis-model can be further simplified by finding an integral manifold on which the voltages E'_d and E'_q remain constant, i.e., the update function equals 0 for these states. Under the assumption of $R_s = 0$, this can be achieved by setting $X'_q = X'_d$ and $T'_{do} = T'_{qo} = \infty$. For further details we refer to [4]. The constant voltages assume the values $E'_q = E'_{q,0}$ and $E'_d = 0$.

Subsequently, we use the following formulation:

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}\left(t, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{y}; \boldsymbol{\lambda}\right) \tag{5.5a}$$

$$\frac{d}{dt}\begin{bmatrix} E'_q \\ E'_d \\ \delta \\ \Delta\omega \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \omega_s \Delta\omega \\ \frac{1}{2H}\left(P_m - E'_d I_d - E'_q I_q - D\Delta\omega\right) \end{bmatrix} \tag{5.5b}$$

$$\begin{bmatrix} I_d \\ I_q \end{bmatrix} = \begin{bmatrix} 0 & -X'_d \\ X'_d & 0 \end{bmatrix}^{-1} \begin{bmatrix} E'_d - V\sin\left(\delta - \theta\right) \\ E'_q - V\cos\left(\delta - \theta\right) \end{bmatrix} \tag{5.5c}$$

with

$$\boldsymbol{x} = \begin{bmatrix} E'_q & E'_d & \delta & \Delta\omega \end{bmatrix}^\top \tag{5.5d}$$

$$\boldsymbol{u} = \begin{bmatrix} E_{fd} & P_m \end{bmatrix}^\top \tag{5.5e}$$

$$\boldsymbol{y} = \begin{bmatrix} \theta & V \end{bmatrix}^\top \tag{5.5f}$$

$$\boldsymbol{\lambda} = \begin{bmatrix} X'_d & H & D \end{bmatrix}^\top. \tag{5.5g}$$

We keep the voltages E'_q and E'_d as part of the state, as it simplifies keeping the following notationally clean. By including all simplifications and directly including the stator equations in $\boldsymbol{f}$, we also can obtain the model used in Part I

$$\frac{d}{dt}\begin{bmatrix} \delta \\ \Delta\omega \end{bmatrix} = \begin{bmatrix} \omega_s \Delta\omega \\ \frac{1}{2H}\left(P_m - D\Delta\omega - \frac{E'_{q,0} V}{X'_d}\sin\left(\delta - \theta\right)\right) \end{bmatrix} \tag{5.6}$$

Load model

Electrical loads could also be modelled as a dynamical system with states, e.g., for electric motors. Alternatively, we can include algebraic load models, i.e., all changes are instantaneous. See, e.g., [5] for examples of load models. Ultimately, we only require a function to compute a complex current injection $\bar{i}$ in the network reference frame[1].

[1]We use the definition that positive injection correspond to injections into the network and negative injections signify withdrawal from the network, i.e., usually loads.

A commonly used load model assumes a dependency of the active and reactive power demand P and Q on the voltage magnitude V

$$\bar{i} = \left(\frac{\bar{s}(V)}{\bar{v}}\right)^* = \left(\frac{P(V) + jQ(V)}{Ve^{j\theta}}\right)^* \tag{5.7}$$

$\bar{s}$ represents the complex power, $\bar{v}$ the complex voltage and the star-superscript indicates the complex conjugate. The voltage dependency can for example assume a power law in V. By assuming, e.g., a linear dependence on V, we obtain a load model that provides a constant current injection. From a methodological point we can combine all algebraic current injection models in

$$\bar{i} = h(\boldsymbol{y}, \boldsymbol{u}; \boldsymbol{\lambda}) \tag{5.8}$$

with $\boldsymbol{y} = \begin{bmatrix}\theta & V\end{bmatrix}^\top$ and $\boldsymbol{u}$ that collects the set-points of the load and $\boldsymbol{\lambda}$ the parameters defining the load characteristics. This provides us with notational consistency with the current injections of the dynamic components.

5.2 Multi-machine simulations - power system DAEs

In order to simulate the dynamics of an entire power system, we need to formulate a system of DAEs

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}; \boldsymbol{\lambda}) \tag{5.9a}$$

$$\boldsymbol{0} = \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{y}(t); \boldsymbol{\lambda}) \tag{5.9b}$$

where $\boldsymbol{x}(t)$ collects all dynamic states, $\boldsymbol{f}$ the corresponding update equations, $\boldsymbol{y}(t)$ all algebraic variables, $\boldsymbol{u}$ all control inputs and $\boldsymbol{\lambda}$ the component and system parameters.

When we restrict the modelling to what is usually referred to as root-mean-square (RMS) models, we neglect transient phenomena in the network and use the phasor representation[2]. In this case, the current flows $\bar{\boldsymbol{i}}^N(t)$ in the network can be computed by the linear algebraic relation

$$\bar{\boldsymbol{i}}^N(t) = \bar{\boldsymbol{Y}}\bar{\boldsymbol{v}}(t) \tag{5.10}$$

with the complex admittance matrix $\bar{\boldsymbol{Y}} \in \mathbb{C}^{n\times n}$ that is based on the connections between the n buses in the network and the line parameters. $\bar{\boldsymbol{v}}(t) \in \mathbb{C}^n$ represents the vector comprised of the complex voltages at the n buses. By imposing the current balance originating from Kirchhoff's law at every bus in the network, we obtain the current balance equation

$$\boldsymbol{0} = \bar{\boldsymbol{i}}^N(t) - \bar{\boldsymbol{i}}^C(t) \tag{5.11}$$

with the current injections $\bar{\boldsymbol{i}}^C(t) \in \mathbb{C}^n$ from the components. At each bus, $\bar{\boldsymbol{i}}^C(t)$ is the sum of the current injections from all connected components; for notational ease, we assume component i is connected at bus i and either one or no component is present at each bus. By summing over multiple components and mapping them to the corresponding bus, other setups can easily be handled. $\bar{\boldsymbol{i}}^C(t)$ is calculated as

$$\frac{d}{dt}\boldsymbol{x}_i = \boldsymbol{f}_i(\boldsymbol{x}_i(t), \bar{v}_i(t), \boldsymbol{u}_i), \quad i = 1, \ldots, m \tag{5.12}$$

$$\bar{i}_i^C = h_i(\boldsymbol{x}_i(t), \bar{v}_i(t)), \qquad i = 1, \ldots, m \tag{5.13}$$

$$\bar{i}_i^C = h_i(\bar{v}_i(t)), \qquad i = m+1, \ldots, m+l \tag{5.14}$$

$$\bar{i}_i^C = 0, \qquad i = m+l+1, \ldots, n. \tag{5.15}$$

[2]By not applying these simplifications, we obtain models to analyse phenomena related to electromagnetic transients, known as EMT models and introduced in [120].

Components 1 to m are dynamic components, e.g., generators, and components $m+1$ to $m+l$ are static components, and at all other buses no component is connected.

The study of DAEs began several decades ago, we refer the interested reader to [121]–[123]. The form of the DAEs that we consider is known as a semi-explicit system of DAEs or an index-1 system of DAEs. That is, since the algebraic variables can be written as

$$\boldsymbol{y} = \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\lambda}) \tag{5.16}$$

as long as the Jacobian $\boldsymbol{g_y}$ is not singular, which we will assume for the remainder of this thesis. This assumption can break, e.g., for bifurcation points, we refer to [5] for a discussion in the power system context. Even though there usually is no analytical expression for $\boldsymbol{y}$, the implicit function theorem ensures its existence, see, e.g., [12]. We silently assume this fact at various places in the following chapters, as it allows the approximation of $\boldsymbol{y}$, e.g., by a series expansion or a Neural Network (NN).

5.3 Solving power system DAEs

The development of solution algorithms for these systems of DAEs has seen continuos research effort and the establishment of commercial programmes for the power system context. Two long-studied approaches are the Partitioned-Explicit (PE) and the Simulataneous-Implicit (SI) solution approach that we will briefly explain in the following, for detailed descriptions we refer to [4], [5], [13]. We proceed with presenting approaches proposed to accelerate these simulations; we place a special focus on Semi-Analytical Solution (SAS)-methods and Machine Learning (ML)-based approaches, as they relate closest to the two subsequent chapters.

5.3.1 Partitioned-Explicit (PE) solution approach

As the name suggests, we partition, i.e., divide, the problem into several sub-problems. The partitioning happens at the interface between the component and the network. This leaves us with a set of problems that describe the component dynamics and another set that contains the network algebraic equations. The two sets are computed in an alternating fashion with the latest available variable of the other problem set. This approach offers great flexibility for the choice of the solution algorithm for each set of problems. Furthermore, each iteration is fast as explicit approximations can be used for the component dynamics. However, the computed solution is not necessarily consistent, it leads to an "interface" error. Repeated alternation within a time step and the use of extrapolation schemes can alleviate this problem at the expense of increased algorithmic complexity and computational cost. The bigger issue is the numerical stability of the explicit approximators. Stiff problems require very small time steps to accurately resolve the dynamics which in turn offsets the fast evaluation speed of the explicit schemes.

5.3.2 Simultaneous-Implicit (SI) solution approach

In the simultaneous-implicit approach, the differential equations are turned into algebraic equations using for example the trapezoidal integration method, here shown for a single component:

$$\boldsymbol{x}(t_1) = \boldsymbol{x}(t_0) + \int_{t_0}^{t_1} \boldsymbol{f}\left(\boldsymbol{x}, \bar{v}, \boldsymbol{u}; \boldsymbol{\lambda}\right) dt \tag{5.17}$$

$$\approx \boldsymbol{x}(t_0) + \frac{h}{2}\left(\boldsymbol{f}\left(\boldsymbol{x}(t_1), \bar{v}(t_1), \boldsymbol{u}; \boldsymbol{\lambda}\right) + \boldsymbol{f}\left(\boldsymbol{x}(t_0), \bar{v}(t_0), \boldsymbol{u}; \boldsymbol{\lambda}\right)\right) \tag{5.18}$$

where $t_1 = t_0 + h$. Together with the algebraic equation for the current balance evaluated at t_1, we obtain a system of implicit algebraic equations. By iteratively updating the variables $\boldsymbol{x}(t_1)$ and $\bar{\boldsymbol{v}}(t_1)$ using, e.g., the Newton-Raphson algorithm, we solve for the desired value, i.e., the states and voltages at the end of the time step. The advantage of this approach is that implicit integration schemes are well suited for stiff problems thanks to their numerical stability. Furthermore, the solution is consistent up to the specified tolerance level, i.e., we by design avoid the interface errors that occur for the PE approach. The drawback of SI approaches is the need to solve a system of implicit equations which can be computationally demanding and time-consuming.

5.3.3 Parallelisation

An important potential for acceleration lies in the ability to parallelise computations. This reduces the "wall-time", i.e., the time that the execution of the program takes. This makes parallelisation particularly important when the un-parallelised computation is too slow for real-time execution. However, the communication and synchronisation between the parallel computations introduces additional computational effort and implementation complexity. We refer to [19] for an extensive coverage of this topic. An important concept in that respect is domain decomposition. A decomposition allows to divide a large problem into several smaller problems that can be executed in parallel. How to decompose a system, e.g., the DAEs, is a modelling choice and depends on problem formulation choice. Prominent decomposition approaches split the problem according to its topology, across the temporal domain or by decomposing numerical operations itself. The authors in [20], [21] summarise recent developments in this area.

5.3.4 Semi-analytical solution (SAS) approaches

The central idea behind SAS-methods is to move some computations away from the run-time/online stage to an offline stage, i.e., pre-computing parts of the solution. The target of these approaches are the dynamic components. Although there exist no closed form solutions for the dynamic state evolutions, as discussed in Part I, good approximations can still prove to be very useful. If these approximations are fast to evaluate and sufficiently accurate, they can accelerate the computation of the time step itself or allow for larger time steps. This aligns exactly with the rationale for using PINNs as a flow approximator (see Section 3.2).

A time-power series-based approximation is the first option as presented in [22]. These approximations, that essentially are based on a Taylor series, are more accurate than a step of a Forward-Euler scheme but do not necessarily achieve the accuracy of higher order Runge-Kutta (RK) methods. However, they can be evaluated by computing a polynomial, which can be significantly faster than multi-stage RK methods. To better account for the non-linearity seen in the state dynamics, Adomian methods [124] can extend the allowable time step size [125].

The resulting approximations can then be used in update schemes that resemble a PE approach (see [125]) or an SI approach (see [22]). In the latter, the complex voltage varies over the time step and this variation is included in the approximation of the component dynamics. We will adopt a similar approach in Chapter 7.

5.3.5 Machine learning-based approaches

A more recent line of work, inspiring also our work, is the application of ML to TDSs, i.e., solving the governing DAEs. The work in [27] introduce PINNs to the case of a single component – it has

been the starting point of our analysis in Part I. A very related approach has been chosen in [126], where the dynamic response of a single component is approximated with DeepONets, an operator learning approach (see Section 3.7).

In [127], the authors also apply DeepONets, now with the objective of learning trajectories resulting from a fault at a pre-specified bus in a multi-machine system. A related method, Fourier Neural Operators, has been used in [128] for learning system dynamics in the frequency domain. The authors in [129] propose a discrete time-stepping scheme (see Section 3.7) for the approximation of the dynamics based on IRK-PINNs similar to Section 3.5.3. They apply the methodology to a three-bus network with classical machine models but raise doubts on the applicability to large-scale power system. The challenge for these methods lies in the high-dimensionality encountered in the power grid as more dimensions usually require increasingly larger datasets in the training process. We will return to this argument in much more detail in Section 6.5 as the scaling of PINNs also faces this challenge.

A slightly different approach is taken in [130], where grid dynamics are expressed as Partial Differential Equations (PDEs). It is not based on the full system of DAEs that we want to consider, but it shows another idea to analyse power system dynamics in large-scale systems with ML.

At this point, we want to stress that our interest in solving power system DAEs originates in the need to perform assessment on the dynamic security of the power system. As [25] reviews, a significant number of ML-based approaches have been suggested for such assessments. However, these approaches do not rely on solving the DAEs explicitly and therefore solve a very different task $\mathcal{T}^3$. Therefore, we do not consider them in them more detail here. In a similarly cautious manner, we must distinguish the task we solve from the term PINNs. As [29] reviews, the term "PINN" has found widespread use in power systems, however, it is not necessarily related the solution of DAEs in multi-machine systems.

5.4 Simulation test cases

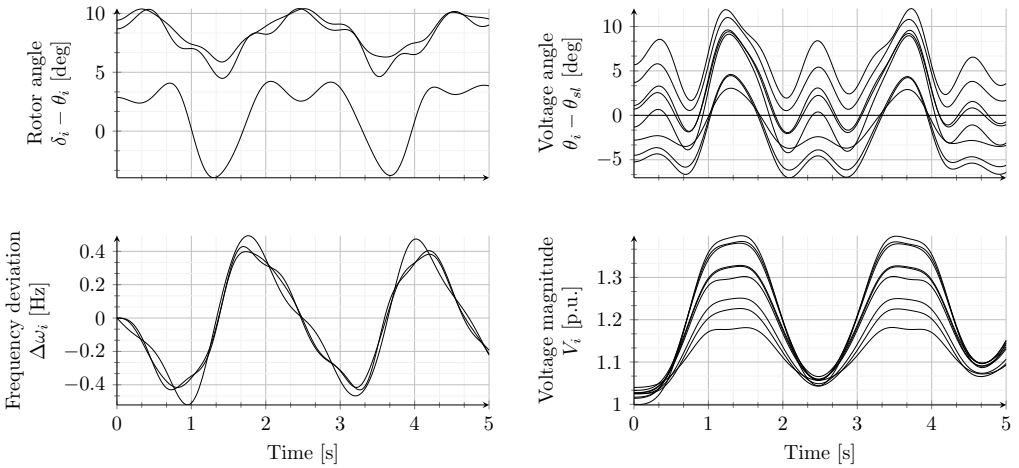
For the purpose of testing the models and algorithms that we develop in Chapters 6 and 7, we will consider three commonly used power system test cases. Our focus lies on the analysis of the numerical characteristics of PINNs in such a setting; we neither aim for an optimised implementation in terms of speed nor try to draw conclusions about the system stability.

To ensure that all learned models share the same model definitions and parameters, we implement a power grid model in Python. This model consists of a classical machine model (5.5) for each generator, the network model (5.5), and constant current loads, i.e., loads with a linear dependence on V in (5.7). For the solution of the resulting system of DAEs, we employ the package *Assimulo* [131] which implements a number of off-the-shelf solvers. We choose a Backward Differentiation Formula (BDF) solver with a tolerance setting of $\varepsilon = 10^{-12}$. It is a multistep method, i.e., not only uses the current state to approximate the integration but also previously computed states. As a comparison, we implement a SI-approach using the trapezoidal rule, see [4] for a detailed description.

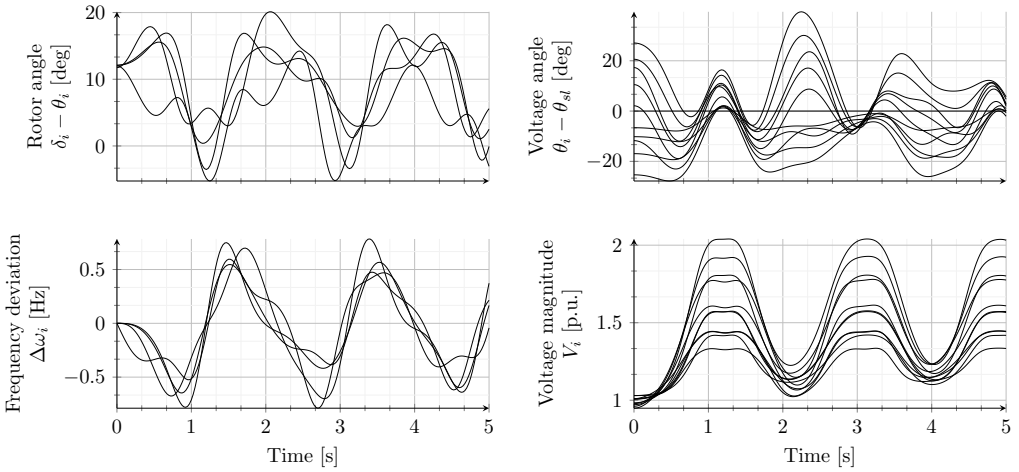
The three tested power systems are a 9-bus system [132] with 3 generators (IEEE 9-bus system), an 11-bus system [3] with 4 generators (Kundur two-area system), and a 39-bus system with 10

[3] See Sections 2.1 and 3.3 for our understanding of the task at hand.

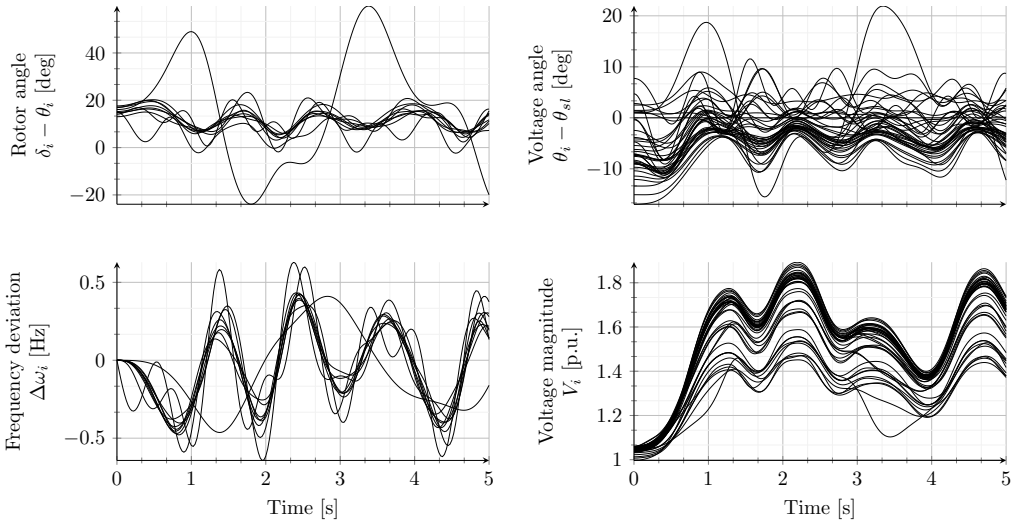
generators [133] (IEEE 39-bus system). The network parameters $\bar{\boldsymbol{Y}}$ stem from the respective reference and the used generator parameters $\boldsymbol{\lambda}$ are listed in Appendix A. For each system, we calculate the steady state solution for the given set-points to obtain the initial condition of the differential states $\boldsymbol{x}(t_0)$. To induce a transient response, we change the control input of a generator in the system and then observe the resulting trajectory. Example of the trajectories that result from this setup and implementation are shown in Figure 5.1.



(a) 9-bus system



(b) 11-bus system



(c) 39-bus system

Figure 5.1: Dynamic response of the three power system test cases to a change of the control input of a generator starting from a steady state at $t = 0$.

CHAPTER 6

Scalability of PINNs in power systems

In this chapter, we investigate the scalability of a Physics-Informed Neural Network (PINN)-based approach for power system dynamics, i.e., the central question of this thesis. We do not aim to provide a definitive answer to the question; our goal is to highlight the trends and characteristics that can enable or prevent scaling PINNs to large multi-machine systems.

The guiding example for this chapter will be a Time-Domain Simulation (TDS) of the dynamic response to a change of a generator control input. The details of the experiment setup are described in Section 6.1. To structure the analysis, we split the assessment of PINNs for this task into two parts: 1) The characteristics at run-time in Section 6.2, i.e., whenever simulations need to be run; 2) the up-front characteristics in Section 6.3 that govern the learning procedure. Being convincing on the former is a necessity for PINNs to offer a benefit over *conventional* methods, i.e., Differential Algebraic Equation (DAE) solution methods that are well-established, such as trapezoidal integration and multistep methods as reviewed in Section 5.3. In contrast, the up-front characteristics determine the feasibility to train such models. Based on these characteristics, we conduct a "break-even" analysis on the total computational cost of PINNs in Section 6.4 as a function of the number of task evaluations. This leads us in Section 6.5 to the "curse of dimensionality" and the limits it might impose on the scalability of PINNs. In Section 6.6, we outline potential solutions to avoid this "curse" – one of which will be treated in the subsequent chapter – before concluding in Section 6.7.

6.1 Experiment setup

We consider the test cases as described in Section 5.4, i.e., a 9-, 11-, and 39-bus systems. To cause a transient response, we change the initial mechanical input $P^0_{m,i}$ of one generator, indexed by i, by η so that the new mechanical power $P^1_{m,i}$ calculates as

$$P^1_{m,i} = P^0_{m,i}(1+\eta). \tag{6.1}$$

We observe this trajectory over 5 s. Section 5.4 shows trajectories that correspond to $\eta = -0.5$ when changing the input of generator $i = 1$, $i = 2$, or $i = 5$ for the 9-bus, 11-bus, or 39-bus system respectively. For all three cases, we simulate datasets for $\eta \in [-0.5; 0.0]$ and $t \in [0.0, 5.0]$ s. In contrast to Chapter 3, we use equally spaced samples for t and η with increments Δt and $\Delta\eta$. We combine them in a dataset consisting of N points

$$\mathcal{D} = \left\{ \left((t^{(1)}, \eta^{(1)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}) \right), \ldots, \left((t^{(N)}, \eta^{(N)}), (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)}) \right) \right\}, \tag{6.2}$$

where the outputs are the differential and algebraic variables $\boldsymbol{x}$ and $\boldsymbol{y}$. By using different discretisation increments of the input domain, i.e., Δt and $\Delta\eta$, we create different dataset sizes. Table 6.1 lists an overview of the simulated datasets. We will refer to them as *scenarios* A to E to simplify the description – scenario A is a setup with sparse data and scenario E is data-rich. The test

Table 6.1: Overview of the scenarios with different training datasets.

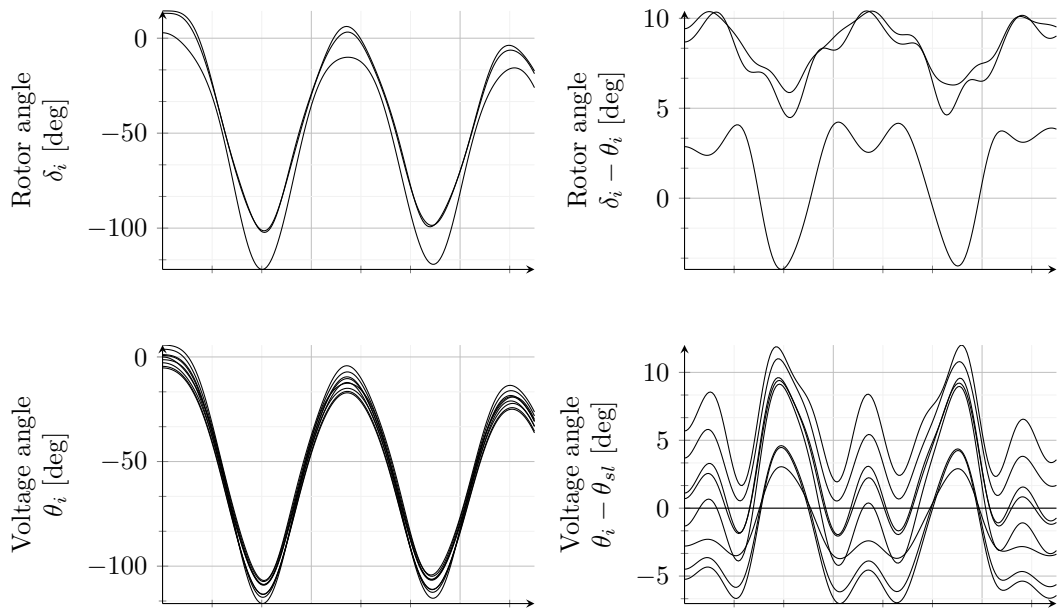
Scenario	Time increment Δt	Control input increment $\Delta\eta$	Dataset size $\lvert\mathcal{D}\rvert$
A	1.0 s	10 %	66
B	0.5 s	10 %	126
C	1.0 s	5 %	121
D	0.5 s	5 %	231
E	0.1 s	1 %	2601
Test	0.005 s	0.25 %	201201

dataset will be used across all scenarios and a validation dataset is formed by using the same increments as for the training dataset but offset by $\frac{\Delta t}{2}$ and $\frac{\Delta\eta}{2}$ in the respective dimension. The initial conditions equal the equilibrium state of the system.

The Neural Network (NN), i.e., the hypothesis class h_θ, takes $\boldsymbol{z}_0 = [t, P^1_{m,i}]$ as inputs and returns the outputs $[\hat{\boldsymbol{x}}^\top, \hat{\boldsymbol{y}}^\top]^\top$. In the present case, $\hat{\boldsymbol{x}}$ consists of the generator states $[E'_q, E'_d, \delta_i, \Delta\omega]^\top$ for each of the generators and $\hat{\boldsymbol{y}}$ collects the network voltage angle and magnitude at all buses $[\boldsymbol{\theta}^\top, \boldsymbol{V}^\top]^\top$.

In the learning formulation, we use the difference between the bus voltage angles $\boldsymbol{\theta}$ and the slack bus, i.e., $\boldsymbol{\theta} - \theta_{\text{sl}}$ as well as the angle difference between the rotor angle and the terminal voltage angle $\delta_i - \theta_i$. In Figure 6.1, the two options are shown. While the absolute angle frame shows the seemingly easier trajectories, this is an effect due to scaling as the variations seen in the relative angles would still need to be resolved for accurate predictions. This choice furthermore is desirable from the learning perspective as the dominating sinusoidal component of the trajectory only needs to be captured in the prediction of the slack bus voltage angle $\hat{\theta}_{\text{sl}}$. This step can be seen as a pre-processing or feature engineering step in the Machine Learning (ML) context, reminiscent of the architectural ideas laid out in Section 3.5. It furthermore aligns well with our understanding of the underlying physics, where voltage angle differences matter for the power flow, rather than the absolute values[1].

[1]For this reason, one could even eliminate the need for a slack bus angle. However, we did not do so as it complicated the implementation.



(a) Absolute angles **(b)** Relative angles

Figure 6.1: Trajectories in an absolute and relative angle frame. Note the difference of the scaling of the y-axes.

The loss function of the data loss $\mathcal{L}_x$, the dt-loss ($\mathcal{L}_{rhs} + \mathcal{L}_{lhs}$), and the physics loss $\mathcal{L}_c$ formulate as

$$\mathcal{L}_x = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \begin{bmatrix} \hat{\boldsymbol{x}} - \boldsymbol{x} \\ \hat{\boldsymbol{y}} - \boldsymbol{y} \end{bmatrix} \right\|_{T,2}^2 \tag{6.3}$$

$$\mathcal{L}_{rhs} = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \begin{bmatrix} \boldsymbol{f}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) - \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \\ \boldsymbol{g}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \end{bmatrix} \right\|_{T,2}^2 \tag{6.4}$$

$$\mathcal{L}_{lhs} = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \begin{bmatrix} \frac{d}{dt}\hat{\boldsymbol{x}} - \frac{d}{dt}\boldsymbol{x} \\ \boldsymbol{0} \end{bmatrix} \right\|_{T,2}^2 \tag{6.5}$$

$$\mathcal{L}_c = \frac{1}{|\mathcal{D}_c|} \sum_{i=1}^{|\mathcal{D}_c|} \left\| \begin{bmatrix} \boldsymbol{f}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) - \frac{d}{dt}\hat{\boldsymbol{x}} \\ \boldsymbol{g}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \end{bmatrix} \right\|_{T,2}^2 \tag{6.6}$$

The ideas are identical to Section 3.5 with the exception that we need to distinguish between differential variables $\boldsymbol{x}$ and algebraic variables $\boldsymbol{y}$. As, by definition, $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}) = 0$, this term does not occur in the formulation of $\mathcal{L}_{rhs}$ and $\mathcal{L}_c$[2]. As of now, we do not have a rigorous method to define an equivalent to the dimensionless state introduced in Section 3.3. Instead, we apply a linear scaling by $\boldsymbol{\xi}$ of the states $[\boldsymbol{x}^\top, \boldsymbol{y}^\top]^\top$ for calculating the norm $||\cdot||_{T,2}$.

$$\boldsymbol{T}\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}\right) = \operatorname{diag}(\boldsymbol{\xi}) \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \tag{6.7}$$

The values of $\boldsymbol{\xi}$ are chosen based on the variation within in the corresponding states. We provide a detailed description in Appendix A.3.4. The total loss is obtained by weighting the loss terms, analogous to Section 3.5

$$\mathcal{L} = \mathcal{L}_x + \alpha_{dt} \left(\mathcal{L}_{lhs} + \mathcal{L}_{rhs}\right) + \alpha_c \mathcal{L}_c. \tag{6.8}$$

[2] $\mathcal{L}_{rhs}$ could be even extended by considering the update functions with $\boldsymbol{x}$, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{x}}$, $\boldsymbol{y}$.

For vanilla NNs, we set $\alpha_{dt} = \alpha_c$, for dtNNs we choose $\alpha_{dt} \neq 0$ and $\alpha_c = 0$, and for PINNs we include all terms, i.e., $\alpha_{dt} \neq 0$ and $\alpha_c \neq 0$. As the distinction between these NN types is primarily relevant during the training process, we will mostly use the term NN when any of the three types could be used. Further details on the experiment setup such as hyperparameter settings can be found in Appendix A.3.4.

6.2 NNs at run-time

This section will demonstrate the characteristics that NN-based solution approaches possess at run-time, compared to other solution methods for power system DAEs. The two metrics, by which we analyse the solvers, are the evaluation speed and the accuracy. For NNs to be an alternative to conventional numerical solvers, we desire a good trade-off between accuracy and run-time. This trade-off depends on solver specific *settings*, hence, we will discuss how to use them to influence the trade-off for NNs and conventional solvers.

Evaluation speed

Let us first consider the run-time in Figure 6.2, where we plot the solution time of each solver, i.e., the run-time, as a function of the prediction time, i.e., the time instance t at which we want to know the state of the system. The run-time is shown on the upper panels and the columns correspond to the different system sizes. The results clearly suggest, that the NN-based predictions can be several orders of magnitude faster than the ones by conventional solvers. The gap, meanwhile, depends on four aspects: 1) the prediction time t, 2) the system size, 3) the setting of the solver, 4) the efficiency of the implementation.

For the Assimulo solver which uses a multistep Backward Differentiation Formula (BDF) scheme and the solver based on the trapezoidal rule (see Section 5.4), longer prediction times lead to longer run-times. In case of the trapezoidal rule, this is directly related to the nearly linearly growing number of iterations. As the BDF solver constitutes a multistep scheme with a variable internal time step size, the relation is slightly more complicated. For very small prediction times, the "starting" of the method always requires a similar amount of function evaluations. Only as we increase the prediction time, we observe the expected run-time increase. The absolute values of the run-time furthermore show a dependence on the system size as each internal pass[3] becomes more costly for larger systems as it is closely related to the evaluation of $\boldsymbol{f}(\cdot)$. If we instead look at the number of passes of each solver (shown in the lower panels), we observe that they are comparable across the three cases. Here we also see the effect of the settings of the solvers, either the tolerance for the BDF solver or the (fixed) time step size h for the trapezoidal rule. Smaller tolerances and smaller time step sizes increase the number of passes and hence the run-time.

For a NN, the trends look very different: The run-time is independent of the prediction time t as we always need only a single pass of the NN. The system size affects the run-time cost, albeit relatively little. This is due to the fact that only some operations in the NN, such as the calculations in the last layer, are directly dependent on the system size. Besides the impact of the system size, the model complexity, which could be considered a "setting" of a NN-based solver, affects the run-time. The run-time scales asymptotically linear $\mathcal{O}(N_L)$ with the number of hidden layers N_L

[3]This measures how often the solver needs to evaluate its internal logic. This can be steps (for the BDF), iterations (for the trapezoidal rule), a forward pass (for the NN).

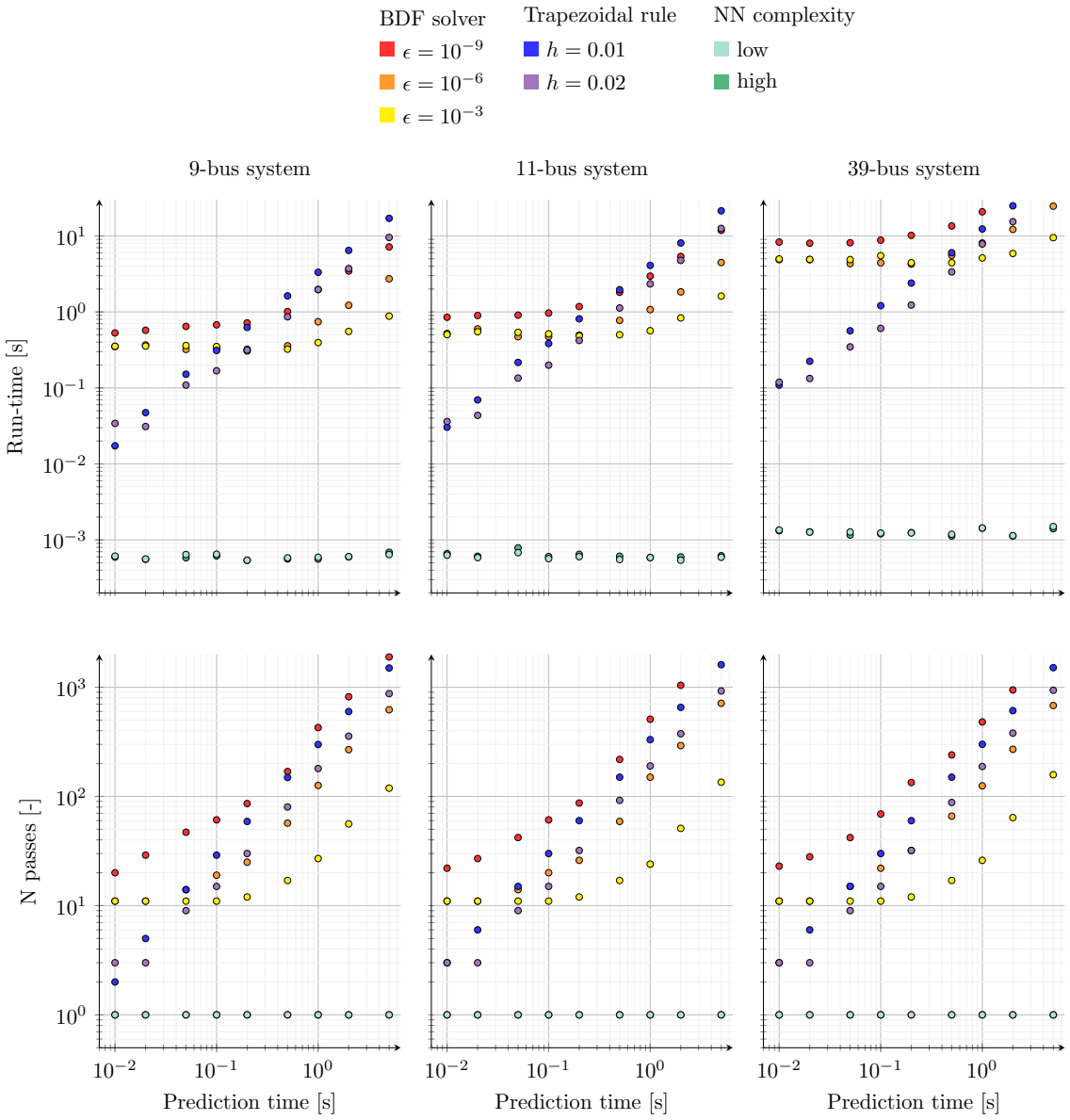


Figure 6.2: Timing of predicting the trajectory

and quadratic ($\mathcal{O}(N_N^2)$) with the number of neurons N_N, i.e., size of the hidden layers, as discussed in Section 3.2.2. The shown results, however, in particular for the 39-bus system, show very little variation, as we have computational overhead of significant scale and the mentioned scaling laws describe asymptotic computational cost.

What we did not cover so far, are the aspects of efficiency of the implementation. We hinted at some influencing factors in Section 3.2, e.g., the hardware, the programming language, and the number of points that are evaluated. As a comprehensive investigation is out of scope of this work, we cannot credibly *quantify* the gap in run-time between NNs and the other solvers. The number of passes, however, gives the most reliable picture as the run-time, in the ideal case, is approximately linearly proportional to the number of function evaluations. The proportionality constant is then the cost of a single pass, which is highly implementation dependent. Given these considerations, we can conclude that NNs are very likely to provide a significant benefit in terms of run-time, the

precise factor will be problem specific.

Accuracy

Naturally, speed alone is not sufficient; the ideal solver would not only be fast but also provide very accurate solutions. Figure 6.3 displays the relationship between these two dimensions for the above cases for the 9-bus system. The three panels show the trade-off between run-time and accuracy for the three solver types (BDF, trapezoidal, and NNs). The regions originate from evaluating a number of points for different prediction times and changes in the control inputs. Appendix B.1 shows the raw data from which we derived Figure 6.3. For the accuracy evaluation, we use the loss of the respective point ℓ (see (3.35)) with the ground-truth being the BDF solver with tolerance $\varepsilon = 10^{-12}$. The shading shall indicate the prediction time t which we analysed in the previous section. A light shading corresponds to a small value of t and the full color to a large value of t. The arrow also points out this trend.

The plots illustrate the characteristics of the present trade-off very clearly. For the BDF solver, increasing the internal tolerance ε allows for a small reduction of the run-time at the expense of less accurate solutions. When using loose tolerances, i.e., larger ε, the accuracy deteriorates quickly for longer prediction times. The built-in error control on the other hand results in maintaining highly accurate, i.e., low loss, even for long prediction times when the internal tolerance is tight – this error control ensures that we usually consider simulations with tight tolerance settings as *ground truth*. If we omit such error control and simply use a fixed time step size, as we do for the trapezoidal rule, the error inevitably accumulates for longer prediction times. In order to maintain sufficient accuracy, we need to reduce the time step size h, but then we require more time steps, thereby increasing the run-time. For NNs, we clearly see that the fast evaluations come at the expense of less accurate solutions. The "setting" for NNs, i.e., the model complexity, shows some effect on the resulting accuracy but not a trend as consistent and predictable as for the other solvers. As we saw before, the prediction time has no significant impact on the run-time.

If we consider the "regions" in Figure 6.3 that the different methods cover, we can characterise their strengths and limitations. An ideal solver, located in the lower left corner, would yield highly accurate and fast solutions. NNs can reach the lower right corner but have limited ability to extend towards more accurate solutions, even when increasing the NN model complexity. For the BDF solver, located in the upper left part of the plot, changing the internal tolerances gives us a lot of control on the horizontal axis, but improving in the vertical direction, i.e., lowering run-time is difficult. For a fixed time-stepping scheme, represented here by the trapezoidal solver, adjusting its settings allows a more "diagonal" shift, but this is limited by divergent solutions for too large time step sizes. The great advantage of the conventional solvers is the flexibility to adjust these solver settings at run-time to adjust to the requirements of the task, which corresponds to choosing the appropriate "solver region". The only way to change the performance of the NN-based solver is to change the entire NN model. This is a consequence of the fact, that the run-time characteristics are entirely determined when training the NN. The described observations hold for the other system size (see Appendix B.1) as well, the prescribed regions will be slightly shifted, for instance, upwards for the 39-bus system due to the longer run-times.

The setting of NNs: Model complexity

Section 6.3 describes *how* to achieve accurate NN models, but we first want to take a closer look

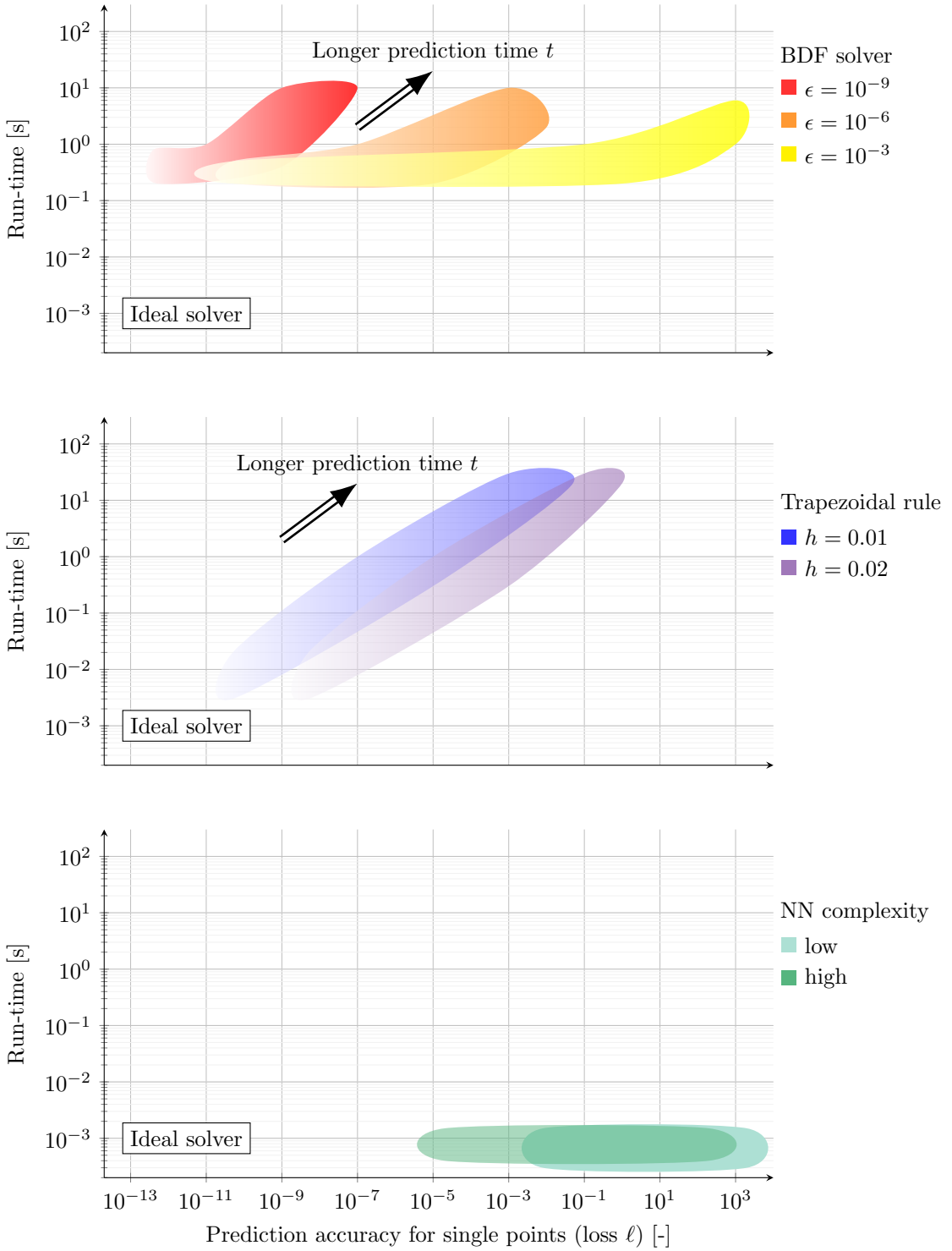


Figure 6.3: Solver regions in the trade-off between run-time and accuracy. Example from the 9-bus system

at the outcome, the trained model. We do so by considering the relation between the NN model complexity and the resulting accuracy, i.e., analysing the Bias Variance Trade-off (BVTO) shown in Figure 6.4. As expected, the generalisation gap increases significantly for higher model complexities as the training loss (empty boxplots) continues to decrease, while the testing loss (coloured boxplots)

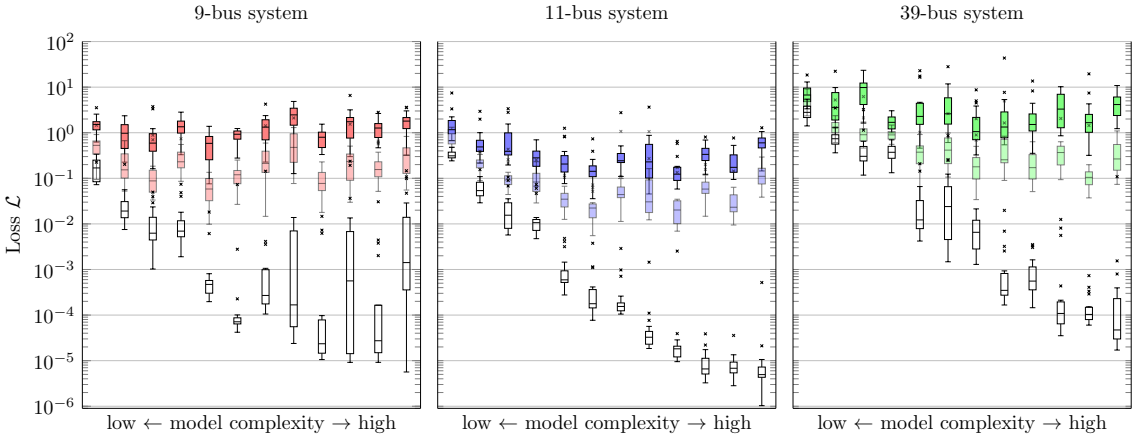


Figure 6.4: Loss values for different NN model complexities and different systems. Training loss (empty boxplot), validation loss (light fill), testing loss (coloured).

and the validation loss (light fill) only slightly decrease. The difference between validation and testing loss strikes as an oddity. It is undesirable as it shows that the validation loss is not an accurate estimate of the testing loss. This is caused by a few bad predictions in the test dataset as we will see in the next section and can be attributed to the dataset characteristics not being similar enough. As we use the squared error of the norm for the loss calculation, we obtain a metric that is very susceptible to outliers, which leads to the above results. To expose this effect, use a performance metric $\mathcal{P}$ instead of the loss $\mathcal{L}_x$. We choose the mean absolute error of the different variable groups for $\mathcal{P}$. When evaluated on the test dataset, more complex models show improved results, i.e., lower mean absolute errors, as shown in Figure 6.5. This illustrates the difficulty of using a single metric, here, the loss $\mathcal{L}_x$ in the dimensionless coordinates, to judge the performance of a model. Furthermore, the absolute error is a metric that is system and scaling independent. The loss $\mathcal{L}_x$, in contrast, depends on the dimensionless states, and thereby on the choice of the scaling factor $\boldsymbol{\xi}$. The different number of states in the three system also complicates the interpretation of the magnitude of the loss $\mathcal{L}_x$; a scaling with $1/\dim([\boldsymbol{x}^\top, \boldsymbol{y}^\top])$ could be included. The interesting insight from considering the MAE is that the errors seen across the different cases are quite comparable for a given model complexity, apart from very low model complexities[4]. This means, that an increase in system size does not automatically require an increase in model complexity to maintain a certain accuracy in terms of absolute errors. For the case of the voltage angle difference $\theta_i - \theta_{sl}$, the NN for the 39-bus system even achieves lower errors than the NN for the 11-bus system. While this is partially an effect of the particular scaling choice, it furthermore hints at the following: Not the system size per se is important for the attainable error magnitude, but the "difficulty" of the task $\mathcal{T}$ – just as we elaborated on in Section 3.3. A quantification of the task difficulty across different systems is of course non-trivial, but the idea of a task-difficulty provides an intuition on how system size affects the necessary model complexity, which we consider the setting of NN-based solver in the trade-off between run-time and accuracy.

[4]In the definition of the model complexity, we do not account for the output layer of the NN. As systems with more states have automatically a larger output layer, they could be considered slightly more complex/expressive. On the other hand, the dimensionality of the latent space does not change, as the output layer only affects the linear mapping from the latent space to the original state space.

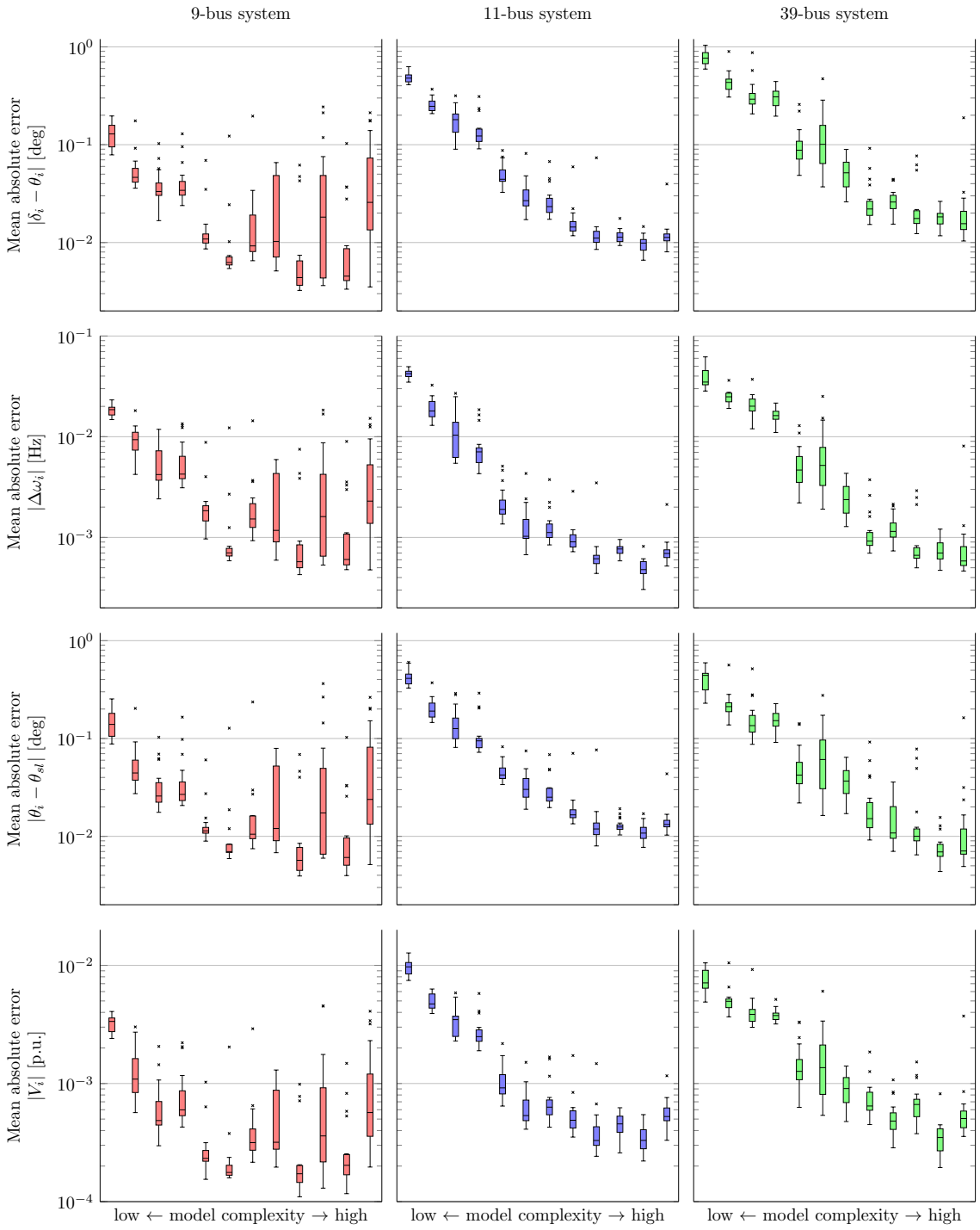


Figure 6.5: Mean absolute errors for different NN model complexity and different systems. The panels along the vertical axis correspond to the same system (same colour). The different variable types are shown in the rows. The mean absolute error is calculated across all buses for the variable type.

6.3 NNs at training time

As the run-time evaluation is performed with a trained NN, all characteristics are decided upon during the training of the NNs. Therefore, we will show in this section, how we can alter the overall accuracy and its characteristics by including domain knowledge, i.e., applying PINNs and how this

relates to the associated up-front cost. The up-front cost can be split into dataset generation cost, NN training cost, and NN assessment cost. Two approaches to alter these costs are the dataset size and the regularisation of the NN with physical knowledge, i.e., using dtNNs and PINNs instead of a vanilla NN. All following experiments utilise the 9-bus system with a NN of intermediate model complexity with $N_L = 2$ layers and $N_N = 32$ neurons per layer. Regarding the dataset size, we test five scenarios (A-E as defined in Section 6.1), starting from a small dataset (A) and then increasing the number of data points in the two input dimensions, i.e., using smaller time increments Δt or smaller control input changes $\Delta\eta$. The insights gained shall help to choose training setups that provide good accuracy while not being overly costly.

Training accuracy

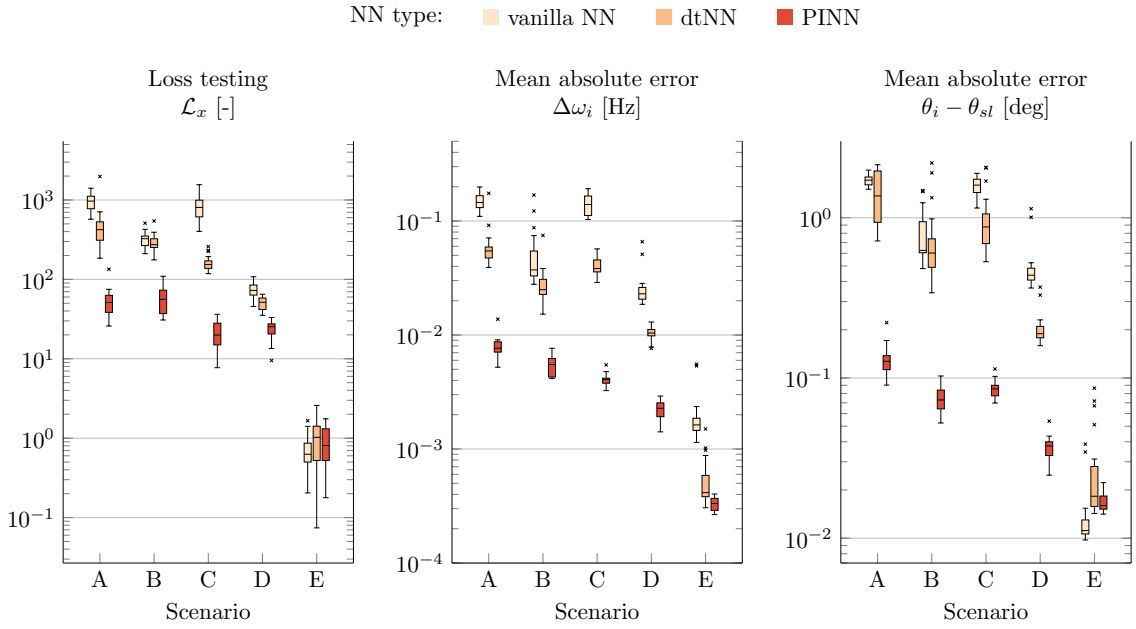


Figure 6.6: Evaluation of training characteristics for different scenarios and NN types (colouring). The three panels show the loss and the mean absolute errors on the frequency deviation and the bus voltage angle differences.

Figure 6.6 summarises the effects of the dataset size and the NN type (vanilla NN, dtNN, PINN) on the accuracy. We observe, that in terms of the loss of the test dataset $\mathcal{L}_x(\mathcal{D}_{\text{Test}})$, shown in left-most panel, additional data points help the overall performance which is to be expected. Also, the addition of physical knowledge helps to improve the performance: dtNNs outperform NNs and PINNs outperform both except for scenario E where all three NN types perform nearly equally well. These results can be confirmed when considering the mean absolute error of the frequency deviation and the voltage angle, shown in the other two panels. When we compare scenario A to B and scenario C to D, which corresponds to adding points along a trajectory (finer Δt), we find the addition of these points to be most beneficial for vanilla NNs. dtNNs show (on a relative scale) less performance improvements as their prediction along the temporal domain is already better due to the additional dt-loss terms. As PINNs already utilise information from collocation points, the addition of more data points has mostly the effect of simplifying the training in terms of convergence as we shall see later on.

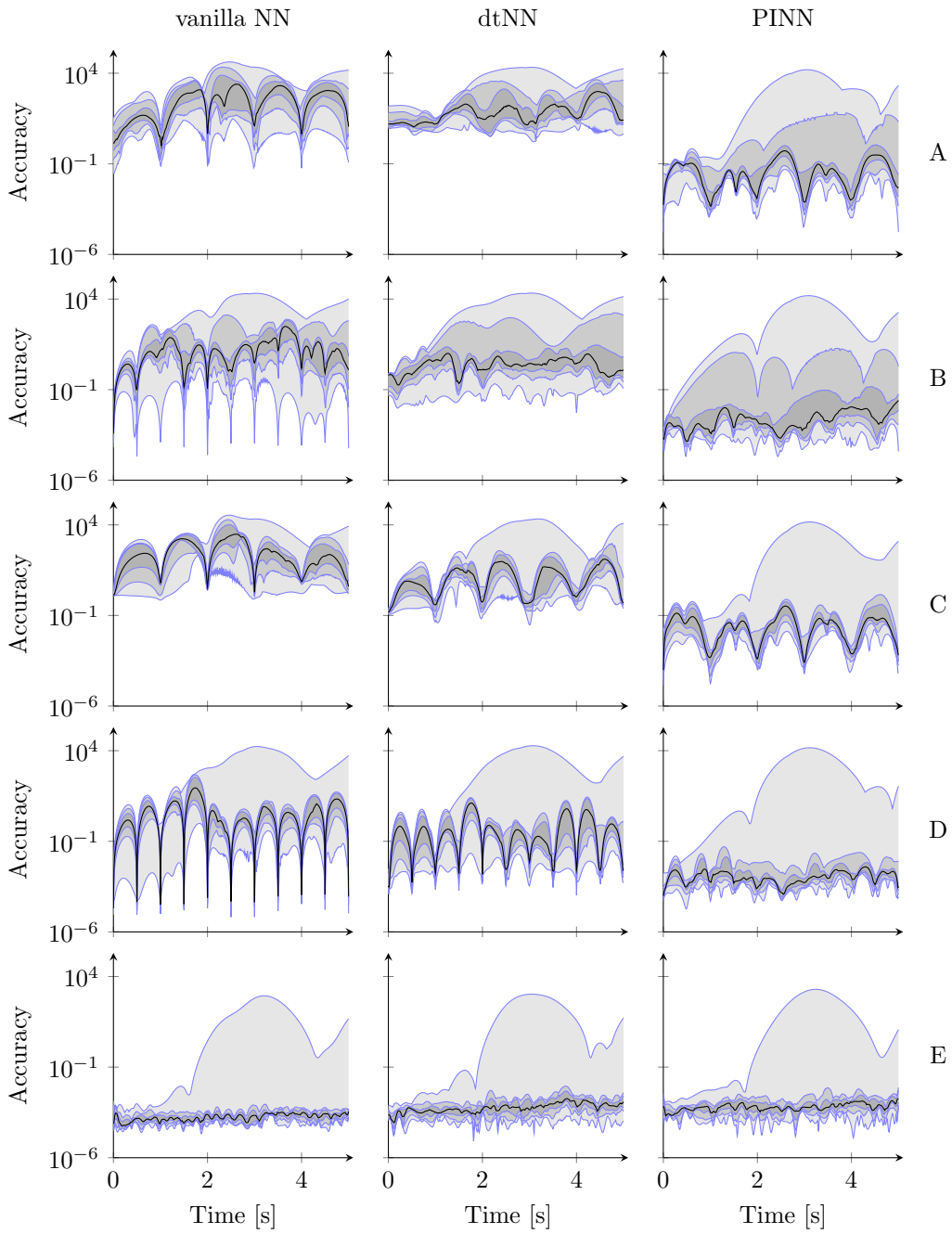


Figure 6.7: Distribution of $\mathcal{L}_x(\mathcal{D}_{\text{Test}})$ as a function of the prediction time t for different NN types (vanilla NN, dtNN, PINN) and scenarios (A-E). The shaded areas correspond to 100%, 80%, 50% of the errors and the black line represents the median. (for the definition of scenarios, see Table 6.1)

These observations can be manifested when considering the loss conditional on the input variables, i.e., on the prediction time t and the change in control input η. The structure of the analysis and plots corresponds to Section 3.6.4 in the single machine context. Figure 6.7 shows the distribution of the loss on the test points as a function of the prediction time t. The shaded areas represent the range between the minimum loss and the maximum loss (light grey), between the 10th and 90th percentile (medium grey), between the 25th and 75th percentile (dark grey), and the median value (black line). For scenarios A-D the "indents" in these distributions signal where the data

points were located, as these points are fitted very well. Only for scenario E, there were enough data points to create a more evenly distributed loss. For scenarios B and D, PINNs also achieve such error characteristics that are less varying. dtNNs improve the absolute loss values but also smoothen the error distributions compared to NNs in most cases, the "indents" remain in other. Across all scenarios and NN types, the large maximum error stands out. This can be explained when considering the error distribution across the other input dimension, namely the change in control input η, shown in Figure 6.8. These plots reveal, that the cause of this high error are

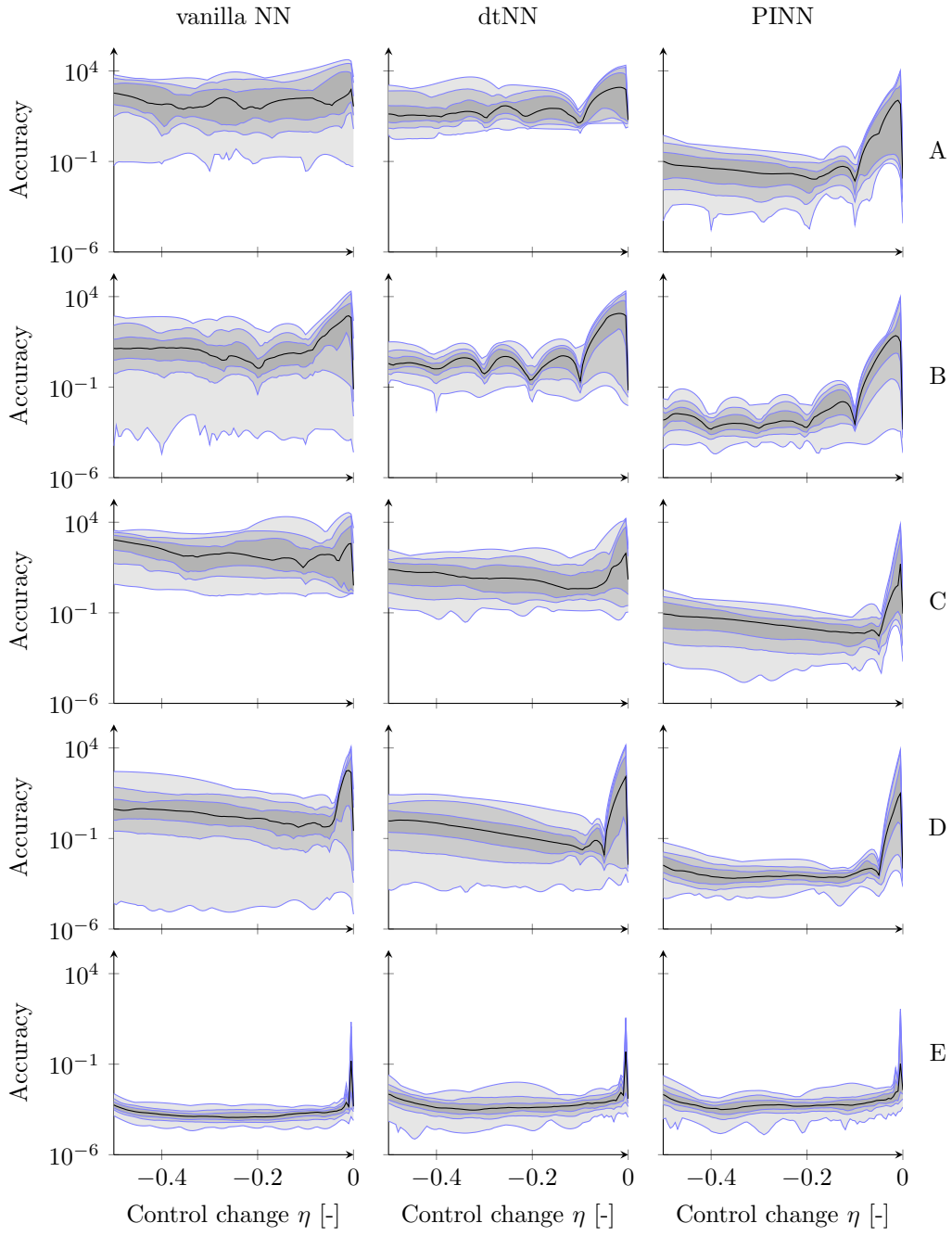


Figure 6.8: Distribution of $\mathcal{L}_x(\mathcal{D}_{\text{Test}})$ as a function of the change in the control input η for different NN types (vanilla NN, dtNN, PINN) and scenarios (A-E). The shaded areas correspond to 100%, 80%, 50% of the errors and the black line represents the median. (for the definition of scenarios, see Table 6.1)

the predictions when there is no change in the control, i.e., when the system remains in steady state. This seems very counter-intuitive, but can be explained by how similar the responses to the change in the control input are. For different values of η, the dynamic response has a similar structure, e.g., the voltage increases for all buses before returning close to the initial state. The constant voltage for the equilibrium case therefore represents a very differently looking pattern which therefore is difficult to predict. For this reason, we obtain an error distribution that is very similar for most values of η. Only between 0 and the next data point, which is at $-\Delta\eta$, we observe high errors, even for the scenario E, that uses the most data points. These observations are of course an artifact of the setup, but they highlight the difficulty when learning dynamics: Simple and obvious solutions, like an equilibrium state, are not necessarily easy to learn. The fact that the equilibrium state lies on the boundary of the input domain aggravates the problem further. It also shows the importance of understanding the model performance $\mathcal{P}$, i.e., the accuracy, beyond the loss $\mathcal{L}_x$ as a single number as we already highlighted in Chapter 3. Such analyses, however, become more intricate the larger the system gets.

Training speed and cost

The improved performance, that we obtain by using more data points or adding additional loss terms, comes at a cost. First, the simulation of the dataset directly incurs a cost. However, we have to distinguish between adding points in the time dimension or the control input dimension. For the former, the cost of additional points is negligible if these points lie on an already included trajectory. In that case, we simply query more points from the solver and since they are based on interpolating an internal solution, more points are nearly "for free". Therefore, the dataset cost C_D in Table 6.2 is the same for scenarios A and B and for scenarios C and D. If we, in contrast, have to simulate an additional trajectory, the total cost increases by the cost of simulating a trajectory. For our case, this means that the dataset cost scales as $\mathcal{O}(1/\Delta\eta)$. The simulation cost for each trajectory C_T then again is a function of the system size, the solver and its settings (the arguments from Section 6.2 apply).

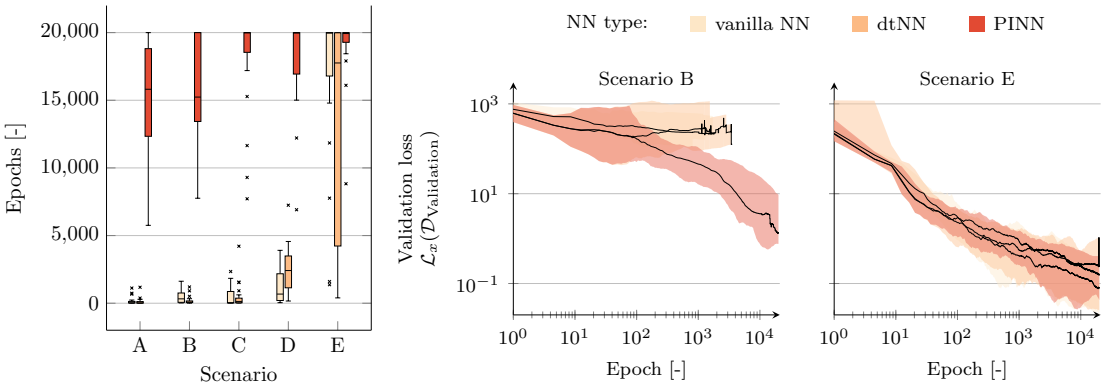
Table 6.2: Overview over computational costs: Cost for the simulation of a trajectory C_T, dataset generation cost C_D, cost per training epoch C_E.

System	C_T	scenario	C_D	C_E [s]		
				NN	dtNN	PINN
IEEE 9	8 s	A	48 s	0.070	0.177	0.406
		B	48 s	0.073	0.187	0.409
		C	96 s	0.073	0.183	0.400
		D	96 s	0.075	0.202	0.420
		E	400 s	0.144	0.471	0.742
Kundur	12 s	E	600 s			
IEEE 39	80 s	E	4000 s			

The other cost is the training itself and here we have to distinguish the effects of larger datasets and the different NN types on the cost per epoch C_E and the number of epochs until we reach

the "best epoch" in terms of validation loss, which we denote by E^*. Table 6.2 reports the cost per epoch C_E. More points lead to a higher cost per epoch C_E, irrespective of the NN type, and also unsurprisingly, dtNNs are more expensive compared to vanilla NNs and PINNs compared to dtNNs. This increase is caused by the calculation $\frac{d}{dt}\hat{\boldsymbol{x}}$ and the evaluation of the update function $\boldsymbol{f}$ necessary for $\mathcal{L}_{lhs}$, $\mathcal{L}_{rhs}$, and $\mathcal{L}_c$. As we evaluate $\mathcal{L}_c$ on the additional collocation points (1000 in this setup), C_E is for all scenarios around $0.2 - 0.3\,\mathrm{s}$ higher than for the dtNN. The exact numbers depend once more on the implementation of $\boldsymbol{f}$ and the entire training procedure as well as the used optimiser parameters, but the trends that more data points and more loss terms increase the cost per epoch are sound and align with the analyses from Section 3.6.5.

The best epoch E^* also shows a dependence on the scenario, i.e., the dataset, and the NN type as Figure 6.9(a) illustrates. For the scenarios with more data points (D, E) the best epoch E^* occurs later on when using vanilla NNs or dtNNs. PINNs, however, require many more epochs for all scenarios. We have to consider, though, that the training process (ideally) continuously improves. Hence, we could stop the training early if we reached a target accuracy to reduce the training cost. We show the improvement of the validation loss in Figure 6.9(b) over the number of epochs, where we can see that PINNs stopped after a few hundred epochs already perform significantly better than vanilla NNs and dtNNs in a small dataset setting such as in scenario B. For a data-rich scenario such as E, this effect disappears.



(a) Best epoch **(b)** Training progress

Figure 6.9: Analysis of the training progress for different scenarios and NN types. The boxplots in (a) represent the results from 20 runs. The shading in (b) illustrates the range of the validation loss across 20 runs as a function of the epoch.

Overall up-front cost

The up-front cost also needs to account for the assessment of the model performance. One part of it is the need for a validation dataset during training. The other is another independent test dataset. The cost of both of them scale just as for the training dataset, i.e., linearly with the number of trajectories. The decision on how many trajectories are necessary to obtain reliable estimates will be another intricate and case dependent matter, that we excluded in our consideration. The detailed investigation of the conditional errors in Figures 6.7 and 6.8 and their interaction with the training setup, i.e., the scenario and the NN type, will be crucial to avoid the need for overly large validation and test datasets.

Given all these considerations, choosing a setup that provides good accuracy while not becoming too computationally expensive will need further research when the different cost drivers are optimised. Conceptually, we can conclude the following though: Vanilla NNs are relying entirely on a sufficiently large dataset. If the generation of such dataset becomes expensive due to long simulation times per trajectory, the use of dtNNs and PINNs becomes more attractive as they reduce the reliance on the dataset, certainly for the training but maybe even for the assessment.

6.4 A view on the total computational costs

We showed in Section 6.2 that NNs have desirable characteristics such as speed and are not susceptible to issues of numerical stability as they only require a single pass. The accuracy will be lower than for conventional solvers like the used BDF-scheme but it can be nonetheless sufficient. The question becomes whether we can train the NNs with a *reasonable* computational effort. For us the dataset generation, the NN training and its assessment, as well as the evaluation carry *computational costs*, measured by the time the computations require. We saw that including physics can help us to be less dependent on the number of simulated data points, hence potentially cutting a significant part of the dataset generation cost. However, it comes at the expense of training for more epochs and with a higher cost per epoch. Striking the balance which approach is favourable will likely be case dependent.

If we abstract, however, from the precise training strategy, we can lump all these costs into the *up-front* cost of a NN. The computational cost that we incur when evaluating the NN is the *run-time* cost – which is independent of the training strategy. This leads to a perspective similar to the fixed-variable-cost curves in economics and manufacturing. The up-front cost constitutes a fixed cost and the run-time cost the variable cost. A conventional solver has usually no significant up-front cost apart from initialisation and compilation – for simplicity we set it to 0. In Figure 6.10 this corresponds to the orange line starting at the origin. The x-axis represents the number of evaluations n of the particular task and the y-axis represents the total cost of it. The NN-based approach instead has relatively high fixed cost due to the expensive training but very low variable cost, yielding a near constant cost curve. Variations in the up-front cost of the NN will alter the cost curve whereas changes in the run-time cost are nearly negligible due to their small size. The dashed lines indicate this. For the conventional solver the change of the slope, i.e., the variable, is obtained by changing the settings such as the tolerance. The cost curves of the two solvers intersect at the point which could be called the *break-even* point for NN-based solvers. Its location depends primarily on the conventional solver's run-time and on the NN's up-front cost as those are the cost drivers for the respective approach. For the presented cases, this break-even point, which defines a critical number of evaluations n_{critical}, lies somewhere on the order between 100 and 1000. This means we would need to reach that number of evaluations for a NN to be an overall more efficient solution method. However, this number is highly sensitive to the run-time cost of the conventional solver and the training cost of the NN, so all results are case and implementation specific. When considering the break-even point as the criterion to judge the viability of an approach, we assume that total computational cost is the deciding factor. If, however, the speed at run-time is crucial, e.g., for real-time decisions, the total computational cost can become a secondary factor. But even in such cases, the break-even analysis hints at the fact that NNs should be used for tasks with a high number of repetitions. To improve the applicability of NNs, we can aim to lower n_{critical} by reducing the up-front cost. Incorporating physics and trying to tailor the training algorithms to

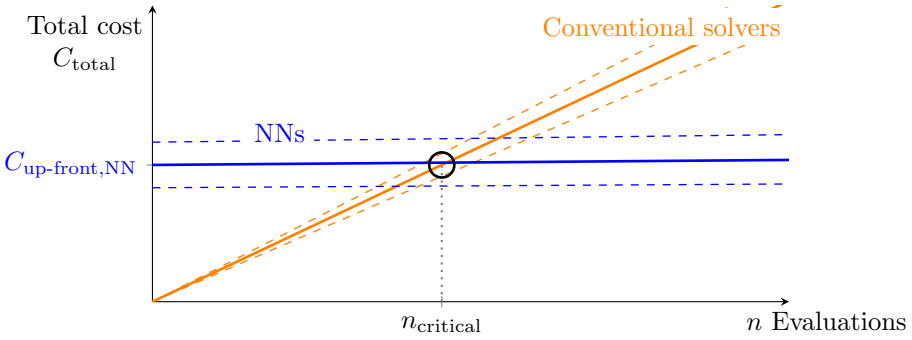


Figure 6.10: Total computation cost curves for NNs and conventional solvers. For NNs, the training (up-front) cost dominate the overall cost, for conventional solver it is the run-time cost. At n_{critical}, the "break-even" point for NNs is reached. The dashed lines indicate a change in the setting of NNs, e.g., model complexity (it can affect the training cost) and conventional solvers, e.g., the tolerance.

the given problems might be part of this effort. Alternatively, problems that are very expensive to solve might be attractive if they can be learned efficiently, i.e., without too high dataset generation or training costs.

6.5 The curse of dimensionality

If we require that the NN provides a benefit from the above total-cost perspective, the applicability of NNs hinges around finding tasks $\mathcal{T}$ for which the number of evaluations n is greater than n_{critical}. This leads us directly to the main challenge of using NNs in the power system context. When we train a NN, we have a particular task in mind. In this chapter, the three tasks were to predict the dynamic response of a power system to the change of the mechanical power at a specified generator for each of the three test power systems. In the setup we implicitly defined all set-points $\boldsymbol{u}$ and parameters $\boldsymbol{\lambda}$ of the system; the latter includes not only the parameters of each component, but also the ones from the power network for its topology and power lines. Furthermore, the assumption of being in steady state at $t = 0$ alleviates the need of picking an initial state $\boldsymbol{x}_0$ that would be of dimension $\mathbb{R}^6$, $\mathbb{R}^8$, and $\mathbb{R}^{20}$ respectively. If we changed any single value of the above, the underlying task for the NN changes.

At this point we run into the infamous "curse of dimensionality", loosely spoken, tasks become "cursed" (practically impossible to solve) when the dimensionality increases too much; combinatorics is a classic example. We will not dive into any rigorous mathematical analysis of this problem as the more intuitive description is sufficient to illustrate our point. We refer the interested reader to [81] as a starting point. In the present case, we can observe this curse from two perspectives. First, through the lens of training a NN: To cover all possible variations of $\boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{x}_0$ we would need to train a NN on all these dimensions and consider them as inputs; here the generation of an adequate dataset would already be practically impossible as Figure 6.11 illustrates. We plot the total number of points N as a function of the points per dimension N_D for problems of different dimensionality $\mathbb{R}^D$. $N = (N_D)^D$ describes this exponential growth. To give an example, $N_D = 2$ would correspond to a dataset where we include all combinations of the maximum or the minimum value in each dimension. Adding a third point in between to might be possible in low dimensional settings, for $D = 15$ though, it leads to more than a 400-fold increase of the total number of points N. Instead of looking at N_D, one could also consider the average distance between points in a normed hypercube

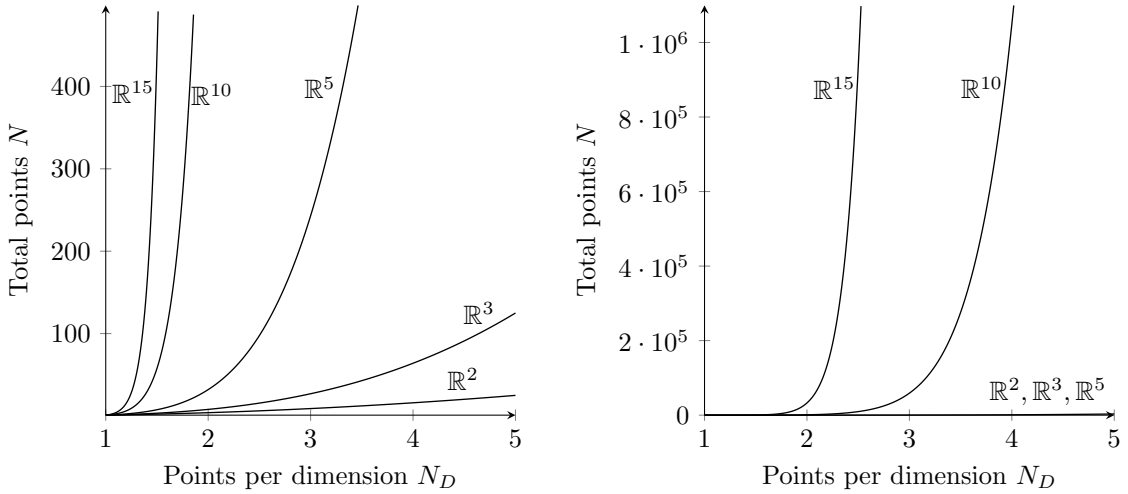


Figure 6.11: Illustration of the curse of dimensionality. The number of points rapidly increases when the average number of points per dimension (in a grid structure) shall be increased.

of dimension D. As we increase D, the points are not any more close to each other which in turn is a problem for the learning process as we do not have control over how the NN interpolates in between these points. Advanced sampling strategies found in Monte-Carlo methods could be used for more "efficient" sampling, but they do not beat the curse of dimensionality. Even in case that we would not need to simulate the used points, i.e., utilising collocation points, there are practical limits to how many points can be effectively evaluated. If evaluating 1 collocation points took on average 1 µs, a dataset based on 5 points for each of 15 dimensions would still require 1 year to run[5] – for a single evaluation.

So if we realistically cannot learn the entire input domain, could we select a lower dimensional subset of the variations and only learn a solution to this subset? This could render the learning task $\mathcal{T}$ with a lower-dimensional input feasible, however, the problem now becomes whether the task occurs sufficiently often or not. Here, the curse of dimensionality strikes again, as the probability of exactly encountering the task, that we trained for, scales inversely with the number of the dimensions which we did not consider as input variables. In power system terms, this corresponds to the question: How often do we encounter the same grid state twice? Or at least a recurring subset of states? In the language of Chapter 3, we could also say that the dimensionality of the flow is cursed, and hence we can only reasonably well approximate certain subsets of that flow but the probability of ending up in this subset is almost 0.

The insight is, that conventional general purpose solvers are incredibly flexible with respect to the task $\mathcal{T}$ they encounter, but they sacrifice speed. NNs in contrast, are comparably inflexible with respect to the task[6], but offer the advantage of fast evaluation. This view resonates with the criteria by which Stott suggested characterising numerical solvers for power systems [13].

6.6 What to do?

In light of the above analyses, we see a few paths forward when it comes to applying NNs to multi-machine simulations, i.e., in a high-dimensional setting. These ideas can already be observed

[5]Without considering parallelisation and any memory or other hardware constraints being ignored.

[6]This "inflexibility" must not be confused with the fact that NNs are a very flexible class of functions to approximate complicated functions.

in other aspects of power systems related tasks, most prominently with power flow computations.

Inductive biases in the architecture of the hypothesis class

Controlling the interpolation in a sparsely populated volume is the fundamental issue behind the curse of dimensionality. The hypothesis class h_θ plays a major role in determining which interpolations are possible. The basic idea is that hypothesis classes which are more restrictive in the possible interpolations can yield better results in sparse settings as the inductive bias is *active*, independent of the evaluation of points, e.g., a collocation point-based inductive bias. The prime example is the use of convolutional neural networks for image recognition. In the power system context, graph neural networks [60] could be a choice as demonstrated for the power flow case [134], [135]. They, by design, incorporate a graph structure, here it could correspond to the power network, and could therefore be better suited for tasks with varying network topologies. The growing field of geometric deep learning [81] studies such inductive biases from a geometric perspective and is worth consulting for inspiration.

Multitask and transfer learning

It might be reasonable to assume that many tasks $\mathcal{T}$ share a common structure. For example, if we changed the control input of a different generator, the resulting trajectories "look" fairly similar for some generators but not for others. By analysing and exploiting these similarities, it might be possible to find more efficient training algorithms and hence reduce the up-front cost of NNs. In ML there exist two fields that are concerned with such setups, namely multitask and transfer learning, see e.g., [49] for an overview. In the former, we train for multiple tasks at the same time. All task predictors are based on a common hypothesis class and then only perform a (small) adjustment to fit each specific task. It can help to generalise and finding powerful representations in the shared NN. Transfer learning instead uses an already trained NN and adapts it to a new task. Here, the use of collocation points could be useful as we would avoid simulating dataset. Both approaches contain a flavour of Reduced-Order Modelling (ROM) techniques as we assume that there is a lower-dimensional manifold with which most of the desired outputs can be explained.

Hybrid approaches

Many conventional algorithms split tasks into several sub-tasks. If the dimensionality of these sub-tasks is feasible to learn and the selected sub-task is often evaluated, NNs can be a promising candidate to accelerate the overall solution time of the algorithm. We will make use of such a hybrid setup in the following chapter by identifying that the prediction of the dynamics of power system components is such a suitable task. It is, in essence, the idea of a Semi-Analytical Solution (SAS)-based solution approach that utilises NNs / PINNs for solving sub-tasks.

Another option for hybrid approaches is to use NNs for the initialisation of conventional algorithms, i.e., to provide a learned "warm-start". We thereby can relax the accuracy requirements on the NN, i.e., fewer data points can be used, while reducing the number of iterations needed for convergence of the conventional algorithm. Potential for acceleration lies in those algorithms where a good initialisation is hard to obtain while important for fast convergence. The author in [136] shows an example of the benefit of learned warm-starts for power flow computations. Using a similar approach could potentially be useful for distributed TDS algorithms such as Parareal [20].

6.7 Concluding remarks

This chapter investigated the general applicability of PINNs to the simulation of power system dynamics in multi-machine systems. The advantage of fast evaluation over long prediction horizons, that we found for the Single-Machine Infinite-Bus (SMIB) system in Chapter 3, can be replicated. Reaching high accuracy, however, becomes more difficult and incurs a significant computational cost. When solving highly repetitive tasks, the run-time speed-ups can justify such up-front cost. The great barrier lies in the curse of dimensionality; the extremely high dimensionality found in power systems appears to prevent the scaling of PINNs into a general purpose solver. Focussing on solving specific tasks seems to be the more promising avenue. Further work on advanced hypothesis classes, training algorithms, and hybrid approaches will be necessary utilise PINNs efficiently on a realistic power system scale.

CHAPTER 7

PINNSim – a PINN-based simulator

The previous chapter illustrated the difficulties that arise, when trying to scale Physics-Informed Neural Networks (PINNs) to larger power system size. Therefore, this chapter explores a hybrid approach between PINNs and conventional Differential Algebraic Equation (DAE) solution algorithms to utilise the ability of PINNs to accurately and rapidly approximate large time step sizes. Following the analysis illustrated in Figure 6.10, PINNs are ideally applied to highly repetitive tasks $\mathcal{T}$. Such tasks can be solved accurately by the same PINN without retraining. In power system dynamics, such mostly "invariant" tasks can be found on the component level. Viewed in isolation, they mostly are exposed to changes of the control inputs $\boldsymbol{u}$ and parameters $\boldsymbol{\lambda}$ and the voltage at the connecting bus or buses $\bar{v}_i$. Hence, if we can include those in the PINN approximation, we can use the same model repeatedly and thereby reach large number of evaluation, i.e., $n > n_{\text{critical}}$ to draw on the advantages of PINNs. Furthermore, we would naturally limit the number of input variables and hence avoid the very high dimensional problems[1]. The key challenge is, how to incorporate the interface between the PINNs and the power grid. The idea of decomposing the system of DAEs, which we explain in Section 7.1, is central to the solution. This leads to the algorithm presented in Section 7.2, with its three elements (a voltage approximation, an update scheme of the voltage approximation, and a state approximator) in Sections 7.3 to 7.5. By combining the three elements, we construct a full DAE simulator in Section 7.6, named *PINNSim* and show its performance in Section 7.7. The results provide a *proof of concept* for this modular simulator. Its scalability does not depend any more on the need to scale PINNs but on their interlacing. As conventional scalable DAE solution algorithms use similar interlacing schemes, we can draw on existing techniques to achieve the scalability of *PINNSim*. Section 7.8 concludes and lays out different areas that are to be improved for developing *PINNSim* into a powerful tool for Time-Domain Simulations (TDSs).

7.1 Decomposing the system of DAEs

We start with the governing system of DAEs introduced in Chapter 5

$$\mathbf{0} = \bar{\boldsymbol{i}}^N(\bar{\boldsymbol{v}}) - \bar{\boldsymbol{i}}^C(\bar{\boldsymbol{v}}, \boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\lambda}) \tag{7.1a}$$

$$\bar{\boldsymbol{i}}^N = \bar{\boldsymbol{Y}}\bar{\boldsymbol{v}} \tag{7.1b}$$

$$\bar{i}_i^C = h_i(\boldsymbol{x}_i, \bar{v}_i, \boldsymbol{u}_i; \boldsymbol{\lambda}_i) \tag{7.1c}$$

$$\frac{d}{dt}\boldsymbol{x}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \bar{v}_i, \boldsymbol{u}_i; \boldsymbol{\lambda}_i). \tag{7.1d}$$

[1] A component model with e.g. 10 states and 3 control inputs and additional parameters poses still a difficult learning task. But we end up with an input dimensionality in the low tens but not hundreds or thousands as it would have been the case otherwise.

To avoid clutter, we drop $\boldsymbol{u}$ and $\boldsymbol{\lambda}$ in the following, but they can be seamlessly included if they shall be altered. Figure 7.1 illustrates the involved quantities graphically. The current balance is a function of the complex voltage $\bar{\boldsymbol{v}}$ given the initial condition $\boldsymbol{x}_0$. However, when we distinguish between $\bar{\boldsymbol{i}}^N$ and $\bar{\boldsymbol{i}}^C$, we observe the following: The network currents are a function of the voltage $\bar{\boldsymbol{v}}$ at all buses whereas the dynamics of the components, and, therefore, their current injections usually depend only on the voltage on a single bus[2]. This "local dependence" allows us to solve all the components' dynamics independently of each other given that we know the complex bus voltages $\bar{\boldsymbol{v}}$. These "sub-problems" are significantly easier to solve and can potentially be parallelised. This leads to the idea behind decomposing the full problem into the network current problem

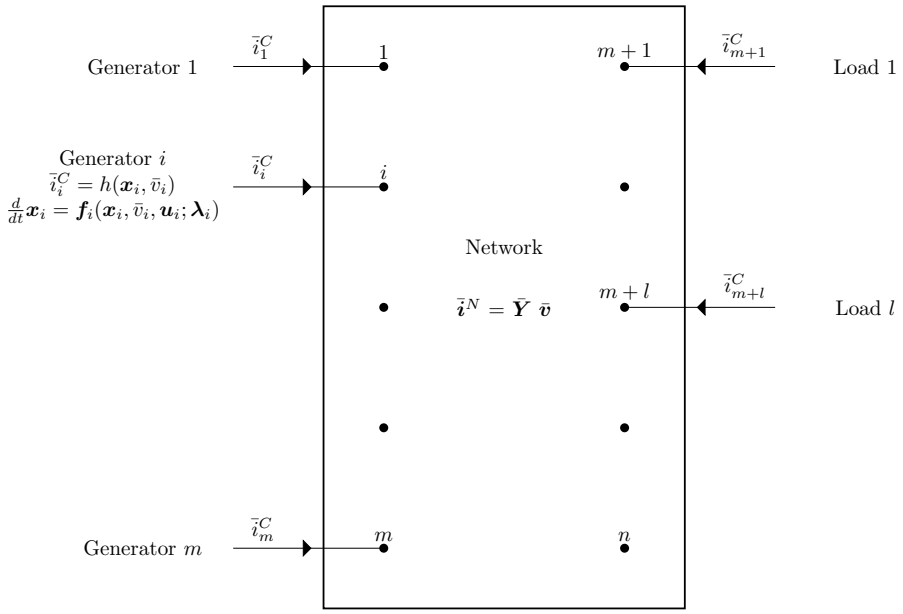


Figure 7.1: Schematic illustration of the decomposition of the power system (adapted from [4, p. 157])

and the m component dynamic problems. The complex voltage $\bar{\boldsymbol{v}}$ serves as the coupling variable and the current balance as the coupling constraint. Whenever we know $\bar{\boldsymbol{v}}$, the sub-problems are straightforward (for $\bar{\boldsymbol{i}}^N$) or at least relatively easy to solve (for $\bar{\boldsymbol{i}}^C$). Such a decomposition underlies many of the approaches presented in Section 5.3. Up to this point the entire formulation is still thought of in a continuous setting, valid at every point along the trajectory. To achieve a tractable solution approach, the Simulataneous-Implicit (SI) solution approach, see Section 5.3.2, introduces at this point a discretisation of the differential equations across a time step of size Δt. Note that we now use Δt instead of h to denote a time step, to avoid the overload of the variable h and align with the notation in [5]. Instead of a discretisation at this point, we follow a SAS-based approach closely related to [22], see Section 5.3.4. The key idea is to express the complex voltage $\bar{\boldsymbol{v}}$ as an explicit and continuous function of time and then approximate it.

[2]Components can also be connected to multiple buses, but the number of buses will remain relatively small. For clarity of the presentation, we only assume components with a single terminal connection.

7.2 Solution approach

We rewrite the above instantaneous equations in an explicit time-dependent form, which requires the integral form for the component dynamics $\boldsymbol{x}_i(t)$

$$\mathbf{0} = \bar{\boldsymbol{i}}^N(\bar{\boldsymbol{v}}(t)) - \bar{\boldsymbol{i}}^C(\bar{\boldsymbol{v}}(t), \boldsymbol{x}(t)) \tag{7.2a}$$

$$\bar{\boldsymbol{i}}^N(t) = \bar{\boldsymbol{Y}}\bar{\boldsymbol{v}}(t) \tag{7.2b}$$

$$\bar{i}_i^C(t) = h_i(\boldsymbol{x}_i(t), \bar{v}_i(t)) \tag{7.2c}$$

$$\boldsymbol{x}_i(t) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}_i(\boldsymbol{x}_i(\tau), \bar{v}_i(\tau))d\tau. \tag{7.2d}$$

We then replace the true voltage $\bar{\boldsymbol{v}}(t)$ by an approximation $\hat{\bar{\boldsymbol{v}}}(t)$.

$$\hat{\bar{\boldsymbol{i}}}^N(t) = \bar{\boldsymbol{Y}}\hat{\bar{\boldsymbol{v}}}(t) \tag{7.3a}$$

$$\hat{\bar{i}}_i^C(t) = h_i(\hat{\boldsymbol{x}}_i(t), \hat{\bar{v}}_i(t)) \tag{7.3b}$$

$$\hat{\boldsymbol{x}}_i(t) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}_i(\boldsymbol{x}_i(\tau), \hat{\bar{v}}_i(\tau))d\tau. \tag{7.3c}$$

We, on purpose, leave out the current balance, as we do not satisfy this constraint unless $\hat{\bar{\boldsymbol{v}}}(t) = \bar{\boldsymbol{v}}(t)$ over the entire time interval. One can think of this as a relaxation of the coupling constraint. Our goal is to iteratively update the voltage approximation $\hat{\bar{\boldsymbol{v}}}^{(k)}(t)$ such that we reduce this mismatch, i.e., tightening the relaxation.

Therefore, the algorithm, depicted in Figure 7.2, starts by estimating an initial voltage profile for each bus. Based on that estimate, we can evaluate the dynamic evolution of all components $\hat{\boldsymbol{x}}_i(t)$ and their current injection $\hat{\bar{i}}_i^C(t)$ into the power grid. At the same time, the network currents $\hat{\bar{\boldsymbol{i}}}^N(t)$ can also be computed. The current injections and the network flows should at all time obey Kirchhoff's current balance law. However, given the estimated voltage profile, this is likely to be not true; there will be a complex current error $\bar{\boldsymbol{\varepsilon}}$ and the objective becomes to minimise the error by changing the estimate of the voltage profile $\hat{\bar{\boldsymbol{v}}}^{(k+1)}(t)$ in the next iteration. To be more precise, we parametrise the approximated voltage profile $\hat{\bar{\boldsymbol{v}}}^{(k)}(t, \boldsymbol{\Xi}^{(k)})$ by a set of parameters $\boldsymbol{\Xi}^{(k)}$ and we achieve a change in the voltage profile by iteratively updating these parameters with $\boldsymbol{\Xi}^{(k+1)} = \boldsymbol{\Xi}^{(k)} + \Delta\boldsymbol{\Xi}^{(k)}$.

This algorithm has two components conceptually similar to [22]: The use of a parametrised voltage approximation $\hat{\bar{\boldsymbol{v}}}(t, \boldsymbol{\Xi}^{(k)})$, which we will discuss in Section 7.3 and a scheme to update this voltage parametrisation that we discuss in Section 7.4. The big difference comes into play at the step where we approximate the state evolution $\hat{\boldsymbol{x}}$ to obtain the current injection $\hat{\bar{\boldsymbol{i}}}^C$ as a function of the parametrised voltage to then compute the current error. Wang, Duan and Sun used in [22] a power-series for the state approximation; instead, we propose the use of PINNs for this task. The rationale for this adaptation follows the discussion in Chapter 3, namely the capability of PINNs to provide fast evaluations while maintaining high accuracy for long time step sizes. Section 7.5 presents the adaptations that we have to make to the learning formulation for PINNs compared to Chapter 3.

We will see that the resulting algorithm is conceptually simple to implement, encourages parallelisation and provides a continuous solution across a time step. It furthermore becomes computational cost-wise largely independent of the dimensionality of the components' state space as it depends on

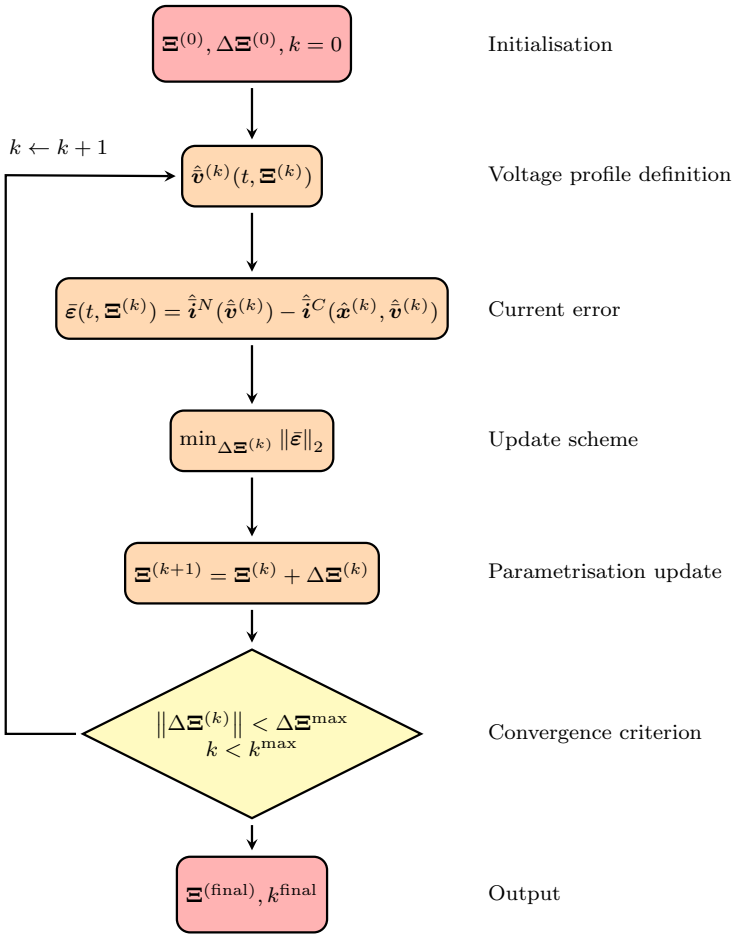


Figure 7.2: PINNSim solution approach for simulating a time step for a system of DAEs.

the size of the PINNs instead[3]. As the above algorithm solves the system of DAEs for a single time step, a conventional time-stepping scheme leads to a full simulator as we outline in Section 7.6. After presenting the different steps of the algorithm, Section 7.7 will show and discuss the factors that influence the accuracy and speed of the resulting solution.

7.3 Voltage parametrisation $\hat{\bar{\boldsymbol{v}}}$

The temporal evolution of the complex voltage $\bar{v}_i(t)$ at bus i could be expressed in multiple ways. We follow [22] and opt for a Taylor series of order r around $\bar{v}_i(t_0)$ in polar form, i.e., for its magnitude $V_i(t)$ and angle $\theta_i(t)$

$$\bar{v}_i(t) = V_i(t)e^{j\theta_i(t)} \approx \left(\sum_{k=0}^{r} V_{k,i}(t - t_0)^k\right) e^{j\left(\sum_{k=0}^{r} \theta_{k,i}(t-t_0)^k\right)} = \hat{\bar{v}}_i(t) \tag{7.4}$$

The coefficients $V_{0,i}, V_{1,i}, \ldots, V_{r,i}$ and $\theta_{0,i}, \theta_{1,i}, \ldots, \theta_{r,i}$ form the parameters which we will later on update to improve the initial estimate. We subsequently refer to this estimate as $\hat{\bar{v}}_i(t, \boldsymbol{\Xi}_i)$. The vector $\boldsymbol{\Xi}_i$ collects all parameters at bus i

$$\boldsymbol{\Xi}_i = \begin{bmatrix} V_{0,i} & \theta_{0,i} & \ldots & V_{r,i}, \theta_{r,i} \end{bmatrix}^\top, \quad \boldsymbol{\Xi}_i \in \mathbb{R}^{2(r+1)}. \tag{7.5}$$

[3]The size will be dependent on the task difficulty as seen in Chapter 3 which in turn will be affected by the dimensionality of the component's state.

We conduct this parametrisation for the voltage at all n buses in the system and collect the corresponding parameters in the vector $\boldsymbol{\Xi}$

$$\boldsymbol{\Xi} = \begin{bmatrix} \boldsymbol{\Xi}_i^\top & \dots & \boldsymbol{\Xi}_n^\top \end{bmatrix}^\top, \quad \boldsymbol{\Xi} \in \mathbb{R}^{2(r+1)n} \tag{7.6}$$

The choice of an appropriate basis function in (7.4) is free, however, a small number of parameters is desirable. For this reason, we for example choose the polar form over the rectangular form as the former extrapolates better as reported in [13], [137]. The benefit of few parameters is, first, that we add fewer dimensions to the learning problem that approximates the components' dynamical responses, see Section 7.5. Second, the optimisation problem that we need to solve later on to update the parameters $\boldsymbol{\Xi}$, see Section 7.4, will be of size $\dim(\Xi) = 2(r+1)n$.

Let us for a moment assume that we knew the evolution of the true voltage $\bar{v}_i$. Our goal is to find a suitable approximation $\hat{\bar{v}}_i$ as defined above. To judge the quality of the approximation, we calculate the residual between the parametrised voltage $\hat{\bar{v}}_i$ and the true voltage $\bar{v}_i$ and apply a norm over the interval

$$\left\| \hat{\bar{v}}_i(t) - \bar{v}_i(t) \right\|, \quad t_0 \le t \le t_0 + \Delta t. \tag{7.7}$$

In practice, we can only evaluate this norm – we choose the L^2-norm – on a finite number of points on the interval

$$\left\| \hat{\bar{v}}_i(t_k, \boldsymbol{\Xi}_i) - \bar{v}_i(t_k) \right\|_2 \quad t_k \in [t_1, \dots, t_s]. \tag{7.8}$$

This evaluation will depend on the number of *query points* s, but its value divided by the number of points should converge when we evenly space the s points and increase their number[4]. This, however, also means, that the number and placement of the s points will determine which set of parameters $\boldsymbol{\Xi}_i$ for a given polynomial order r is considered the best approximation determined in the following minimisation problem, a classical least-square problem

$$\min_{\boldsymbol{\Xi}_i} \left\| \hat{\bar{v}}_i(t_k, \boldsymbol{\Xi}_i) - \bar{v}_i(t_k) \right\|_2, \quad t_k \in [t_1, \dots, t_s]. \tag{7.9}$$

In the following, we illustrate these considerations by comparing the outcome of the approximations of the voltage magnitude $|\hat{\bar{v}}_i|$ as a function of s and r. In Figure 7.3, we show voltage profiles of increasing order r for two time interval lengths ($\Delta t = 0.5\,\mathrm{s}$ and $\Delta t = 2\,\mathrm{s}$) that is fitted using a large number of query points s. These fits can be considered the near optimal parametrisation when using the L^2-norm as the metric. For $\Delta t = 0.5$, it is clear that already for $r \ge 5$, the approximation is visually indistinguishable from the exact solution. For the larger time interval, the voltage profile is more complicated, hence, we need a significantly higher order approximation. The least-square problem (7.9) requires $s \ge r + 1$ to be not under-determined. For a linear fit, i.e., $r = 1$, we need at least two points to determine a unique fit. For three or more (distinct) points the system will be over-determined. By increasing the number of "excess" points $s - (r+1)$, the objective value in (7.9) will converge to the value that corresponds to the optimal parametrisation.

In Figure 7.4 we show this converging behaviour. The error on the y-axis is the value of (7.7) evaluated with a discretisation of $0.001\,\mathrm{s}$. By increasing the number of excess points $s - (r+1)$ in the optimisation problem (7.9), we see the expected converging behaviour. We can furthermore conclude that for more than 5 excess points the error does barely change.

[4]We essentially approximate the temporal integral over the squared residual similar to a Riemann-sum that was also used for the integration of an Ordinary Differential Equation (ODE) in Section 2.2.

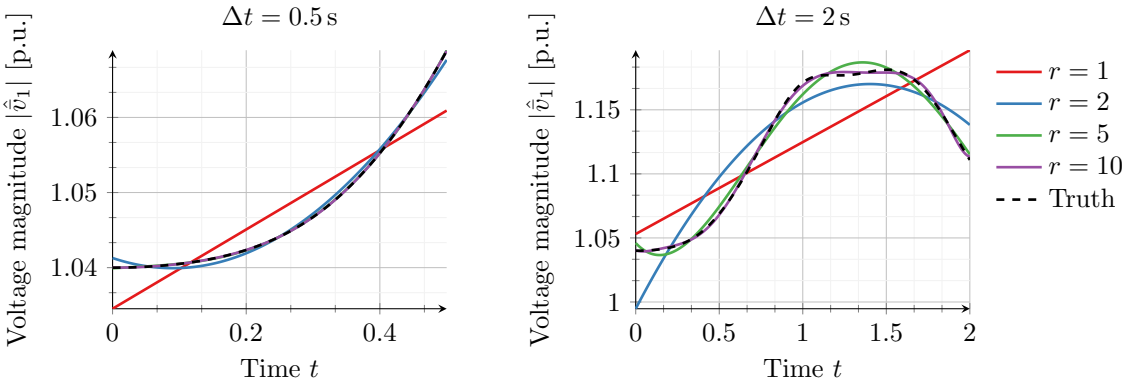


Figure 7.3: Quality of the fit of $\hat{\bar{v}}_i$ for increasing r with $s = (r+1) + 50$.

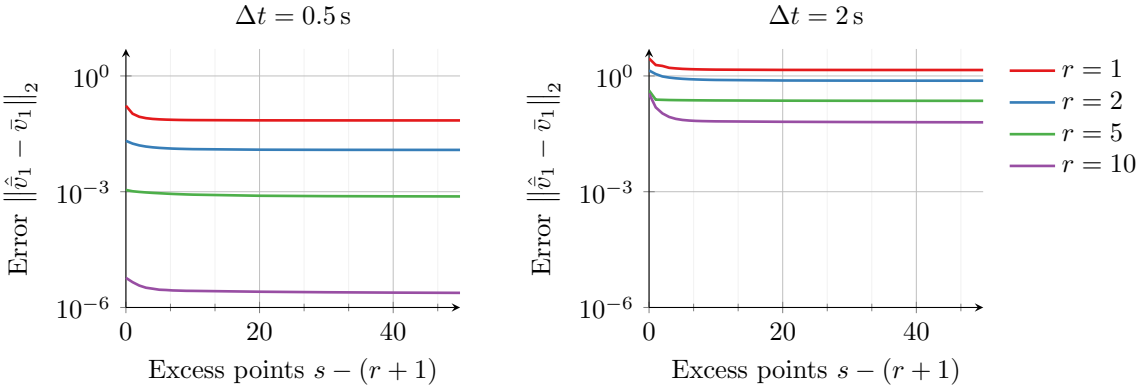


Figure 7.4: Convergence of $\left\|\hat{\bar{v}}_i - \bar{v}_i\right\|_2$ for increasing r and $s - (r+1)$

The above analyses assumed that we knew the true voltage profile. This will not be the case any more in the following; instead, we choose the functional form of the voltage approximations $\hat{\bar{v}}_i$, here a Taylor series of order r, we fix this decision and subsequently only adapt the parameters $\boldsymbol{\Xi}$. The insights from above are, nonetheless, helpful to understand how the parameters r and s influence the approximation accuracy. We will apply a similar optimisation in the subsequent section.

7.4 Update of the voltage parametrisation Ξ

In this step, we "solve" the constructed problem by finding the voltage parametrisation $\boldsymbol{\Xi}$ that leads to the best agreement of the current balance across the interval and all buses. For the exact solution of the DAEs, the current balance must hold across the entire time interval which we write as

$$\begin{bmatrix} \Re\left(\bar{\boldsymbol{i}}^N(\bar{\boldsymbol{v}}(t)) - \bar{\boldsymbol{i}}^C(\bar{\boldsymbol{v}}(t), \boldsymbol{x}(t))\right) \\ \Im\left(\bar{\boldsymbol{i}}^N(\bar{\boldsymbol{v}}(t)) - \bar{\boldsymbol{i}}^C(\bar{\boldsymbol{v}}(t), \boldsymbol{x}(t))\right) \end{bmatrix} = \mathbf{0}, \quad \forall t \in [t_0, t_0 + \Delta t] \tag{7.10}$$

$$\bar{i}_i^C(t) = h_i(\boldsymbol{x}_i(t), \bar{v}_i(t)) \tag{7.11}$$

$$\boldsymbol{x}_i(t) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}_i(\boldsymbol{x}_i(\tau), \bar{v}_i(\tau)) d\tau. \tag{7.12}$$

The voltage approximation $\hat{\boldsymbol{v}}(t, \boldsymbol{\Xi})$ introduces a complex error $\bar{\boldsymbol{\varepsilon}}(t, \boldsymbol{\Xi}; \boldsymbol{x}_0)$ as a function of the voltage parameters $\boldsymbol{\Xi}$ and time t for a given initial condition $\boldsymbol{x}_0$

$$\bar{\boldsymbol{\varepsilon}}(t, \boldsymbol{\Xi}; \boldsymbol{x}_0) := \begin{bmatrix} \Re\left(\hat{\bar{\boldsymbol{i}}}^N(\hat{\boldsymbol{v}}) - \hat{\bar{\boldsymbol{i}}}^C(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}})\right) \\ \Im\left(\hat{\bar{\boldsymbol{i}}}^N(\hat{\boldsymbol{v}}) - \hat{\bar{\boldsymbol{i}}}^C(\hat{\boldsymbol{v}}, \hat{\boldsymbol{x}})\right) \end{bmatrix} \tag{7.13}$$

$$\hat{\bar{i}}_i^C(t, \boldsymbol{\Xi}_i; \boldsymbol{x}_{0,i}) = h_i(\hat{\boldsymbol{x}}_i, \hat{\bar{v}}_i) \quad \forall i \in \mathcal{N} \tag{7.14}$$

$$\hat{\boldsymbol{x}}_i(t, \boldsymbol{\Xi}_i; \boldsymbol{x}_{0,i}) = \boldsymbol{x}_{0,i} + \int_{t_0}^{t} \boldsymbol{f}_i(\hat{\boldsymbol{x}}_i, \hat{\bar{v}}_i, \boldsymbol{u}_i) d\tau. \tag{7.15}$$

The goal becomes to minimise a norm of this current error $\bar{\boldsymbol{\varepsilon}}$ over the interval $[t_0, t_0 + \Delta t]$

$$\min_{\boldsymbol{\Xi}} \left\| \bar{\boldsymbol{\varepsilon}}(t, \boldsymbol{\Xi}; \boldsymbol{x}_0) \right\|, \quad t_0 \leq t \leq t_0 + \Delta t. \tag{7.16}$$

To render the above optimisation problem tractable, we choose to query the current error $\bar{\boldsymbol{\varepsilon}}(t, \boldsymbol{\Xi}; \boldsymbol{x}_0)$ at s points and use the L^2-norm as the metric

$$\min_{\boldsymbol{\Xi}} \left\| \boldsymbol{\rho}(\boldsymbol{t}, \boldsymbol{\Xi}, \boldsymbol{x}_0) \right\|_2. \tag{7.17}$$

The query points are collected in $\boldsymbol{t} = [t_1, \ldots, t_s]$ and their evaluation on the current error $\bar{\boldsymbol{\varepsilon}}$ leads to the residual $\boldsymbol{\rho}$

$$\boldsymbol{\rho}(\boldsymbol{t}, \boldsymbol{\Xi}, \boldsymbol{x}_0) = \begin{bmatrix} \bar{\boldsymbol{\varepsilon}}(t_1, \boldsymbol{x}_0, \boldsymbol{\Xi}) \\ \vdots \\ \bar{\boldsymbol{\varepsilon}}(t_s, \boldsymbol{x}_0, \boldsymbol{\Xi}) \end{bmatrix}, \quad R \in \mathbb{R}^{2ns}. \tag{7.18}$$

We solve this non-linear least-square problem by expanding the residual $\boldsymbol{\rho}$ in a Taylor-series

$$\boldsymbol{\rho}(\boldsymbol{\Xi} + \Delta\boldsymbol{\Xi}) = \boldsymbol{\rho}(\boldsymbol{\Xi}) + \frac{\partial}{\partial \boldsymbol{\Xi}} \boldsymbol{\rho}(\boldsymbol{\Xi}) \Delta\boldsymbol{\Xi} + \mathcal{O}\left((\Delta\boldsymbol{\Xi})^2\right), \tag{7.19}$$

and solve for the voltage parametrisation update $\Delta\boldsymbol{\Xi}$ that eliminates the new residual when assuming a linear dependency on $\boldsymbol{\Xi}$, i.e., $\boldsymbol{\rho}(\boldsymbol{\Xi} + \Delta\boldsymbol{\Xi}) = 0$ and $\mathcal{O}\left((\Delta\boldsymbol{\Xi})^2\right) \approx 0$. This constitutes the Newton-Raphson algorithm and leads to the final formulation which we implement:

$$\min_{\Delta\boldsymbol{\Xi}^{(k)}} \left\| J \Delta\boldsymbol{\Xi}^{(k)} - \boldsymbol{\rho} \right\|_2 \tag{7.20}$$

$$J = \begin{bmatrix} \frac{\partial \bar{\boldsymbol{\varepsilon}}(t_1, \boldsymbol{x}_0, \boldsymbol{\Xi}^{(k)})}{\partial \boldsymbol{\Xi}^{(k)}} \\ \vdots \\ \frac{\partial \bar{\boldsymbol{\varepsilon}}(t_s, \boldsymbol{x}_0, \boldsymbol{\Xi}^{(k)})}{\partial \boldsymbol{\Xi}^{(k)}} \end{bmatrix}, \quad J \in \mathbb{R}^{2sn \times 2rn} \tag{7.21}$$

$$\boldsymbol{\rho} = \begin{bmatrix} \bar{\boldsymbol{\varepsilon}}(t_1, \boldsymbol{x}_0, \boldsymbol{\Xi}^{(k)}) \\ \vdots \\ \bar{\boldsymbol{\varepsilon}}(t_s, \boldsymbol{x}_0, \boldsymbol{\Xi}^{(k)}) \end{bmatrix}, \quad \boldsymbol{\rho} \in \mathbb{R}^{2sn \times 1}. \tag{7.22}$$

By applying an iterative procedure, we aim to converge to the optimal voltage parametrisation $\boldsymbol{\Xi}^{(*)}$ for the given choice of r and s. These iterative update steps are fully determined if $\operatorname{rank}(J) = 2(r+1)n$, hence we will from here on always require $s \geq r + 1$ to avoid under-determined systems[5].

[5] Under the assumptions that $\boldsymbol{g_y}$ is not singular (the entire approach requires it), the s points are distinct, i.e., no coinciding points, and that the component current injection $\hat{\bar{\boldsymbol{i}}}^C$ are not independent of the voltage parametrisation, $s = r + 1$ leads to a non-singular Jacobian.

The Jacobian J contains a lot of structure, which becomes apparent when considering a single query point l

$$\frac{\partial \bar{\boldsymbol{\varepsilon}}(t_l, \boldsymbol{x}_0, \boldsymbol{\Xi}^{(k)})}{\partial \boldsymbol{\Xi}^{(k)}} = \frac{\partial}{\partial \boldsymbol{\Xi}^{(k)}} \begin{bmatrix} \Re\left(\hat{\bar{\boldsymbol{i}}}^N(t_l, \boldsymbol{\Xi}^{(k)}) - \hat{\bar{\boldsymbol{i}}}^C(t_l, \boldsymbol{\Xi}^{(k)}; \boldsymbol{x}_0)\right) \\ \Im\left(\hat{\bar{\boldsymbol{i}}}^N(t_l, \boldsymbol{\Xi}^{(k)}) - \hat{\bar{\boldsymbol{i}}}^C(t_l, \boldsymbol{\Xi}^{(k)}; \boldsymbol{x}_0)\right) \end{bmatrix} \tag{7.23}$$

$$= \frac{\partial}{\partial \boldsymbol{\Xi}^{(k)}} \begin{bmatrix} \Re\left(\hat{\bar{\boldsymbol{i}}}^N(t_l, \boldsymbol{\Xi}^{(k)})\right) \\ \Im\left(\hat{\bar{\boldsymbol{i}}}^N(t_l, \boldsymbol{\Xi}^{(k)})\right) \end{bmatrix} - \frac{\partial}{\partial \boldsymbol{\Xi}^{(k)}} \begin{bmatrix} \Re\left(\hat{\bar{\boldsymbol{i}}}^C(t_l, \boldsymbol{\Xi}^{(k)}; \boldsymbol{x}_0)\right) \\ \Im\left(\hat{\bar{\boldsymbol{i}}}^C(t_l, \boldsymbol{\Xi}^{(k)}; \boldsymbol{x}_0)\right) \end{bmatrix}. \tag{7.24}$$

The first term corresponds to the Jacobian of the network currents $\bar{\boldsymbol{i}}^N$, which yields a large but sparse matrix which can be explicitly formulated as a function of the voltage parametrisation $\boldsymbol{\Xi}$ and time t. The other term represents the Jacobian on the component currents $\bar{\boldsymbol{i}}^C$. At this point, the decomposition idea comes back into play, as the components usually only connect to a single bus. This in turn means that we only obtain non-zero entries in the Jacobian at buses, where a component is connected.

However, to evaluate this Jacobian we need to solve the integrals in (7.15) for the state evolution of the components. Utilising a differentiable ODE solver, e.g., torchdiffeq [138] or DifferentialEquations.jl [139], this is comparably slow but possible thanks to the adjoint method, see descriptions in [118], [119]. When using such a differentiable ODE solver, all error that we introduced so far is related to the quality of the voltage parametrisation as we discussed in Section 7.3. By using a sufficiently accurate voltage parametrisation, i.e., a large enough r, we should be able to bring the current mismatch $\bar{\boldsymbol{\varepsilon}}$ close to zero over the entire time interval.

However, to make this approach attractive from a speed perspective, we have to add another approximation. We replace the exact ODE solver by a fast approximation of its solution, i.e., utilise a flow approximation. As we discussed in Chapter 3, an explicit RK scheme is a potential scheme, however, the accuracy might limit the time step size. Instead, we apply a PINN-based flow approximator $\hat{\Phi}_i$ for each component i to approximate their trajectory $\hat{\boldsymbol{x}}_i$ and then compute the current injection $\hat{\bar{i}}_i^C$. Evaluating the Jacobian $\partial \hat{\bar{i}}_i^C / \partial \boldsymbol{\Xi}^{(k)}$ in either case (RK4 or NN) requires only two backward passes, one each for real and imaginary part of the current injection, a simple operation with Automatic Differentiation (AD).

7.5 Approximation of the component dynamic $\hat{\boldsymbol{x}}$

Here, we consider the approximation of the dynamics of a component, that are described by

$$\boldsymbol{x}_i = \boldsymbol{x}_{0,i} + \int_{t_0}^{t} \boldsymbol{f}_i(\boldsymbol{x}_i, \bar{v}_i, \boldsymbol{u}_i). \tag{7.25}$$

In its essence the question of finding a good approximation $\hat{\boldsymbol{x}}_i$ relates back to the approaches presented in Chapter 3. The only adaptation and extension we need, is to include the voltage parametrisation $\boldsymbol{\Xi}_i$ in the input $\boldsymbol{z}_0$ vector. Whereas we assumed an infinite bus, i.e., a constant voltage at the connecting bus of the component, in Chapter 3, incorporating $\boldsymbol{\Xi}_i$ allows for a dynamic bus formulation as suggested in [22]. The hypothesis class for the PINN becomes

$$\hat{\boldsymbol{x}}_i^{NN} = \boldsymbol{x}_{0,i} + \Delta t \, \mathrm{NN}_i(\boldsymbol{z}_0) \tag{7.26}$$

$$\boldsymbol{z}_0 = \begin{bmatrix} t & \boldsymbol{x}_{0,i} & \boldsymbol{u}_i & \boldsymbol{\Xi}_i \end{bmatrix}. \tag{7.27}$$

We use the residual formulation (3.31) with the consistent initial condition (Residual A) to recover the desirable property of numerical convergence. Reducing the time step size should therefore improve the accuracy later on. A non-trivial part is the choice of input domain. For an ideal solver, this domain spans a large part of the domain of the phase space and the possible voltage parametrisations. We apply an annular sampling for the initial state and linear sampling for time. For the voltage parametrisation, we use a uniform sampling $\mathcal{U}(\min, \max)$ within a box defined by an upper $\overline{\boldsymbol{\Xi}}_i$ and lower $\underline{\boldsymbol{\Xi}}_i$ limit of the parameters $\boldsymbol{\Xi}_i$.

$$\boldsymbol{\Xi}_i \sim \mathcal{U}(\underline{\boldsymbol{\Xi}}_i, \overline{\boldsymbol{\Xi}}_i) \tag{7.28}$$

Choosing these limits is a balancing act between learning too much or too little. In the former the learning task $\mathcal{T}$ is more difficult, which requires larger models and datasets for the same accuracy. In the latter, the accuracy might be low for inputs around or outside the limits which would harm the entire algorithm's accuracy.

To ensure that the resulting trajectory prediction lies within the training domain, a check after convergence of the voltage update algorithm should be performed. During the update of the voltage parametrisation, we may allow it to use predictions beyond the training domain as long as the final parametrisation lies within the training domain. Such a check becomes especially important for cases such as short-circuits or line outages that lead to large voltage deviations outside of normal or desired operating ranges.

7.6 PINNSim - the full algorithm

We combine the previous sections in Algorithm 1 to solve for the evolution of the approximated voltage $\hat{\boldsymbol{v}}$ and the approximated states $\hat{\boldsymbol{x}}$ over a time step of size Δt. It corresponds to the illustration in Figure 7.2. We need to provide the initial conditions of the states $\boldsymbol{x}_0$ and the query points $\boldsymbol{t}$ to the algorithm. Furthermore, we need to set $\boldsymbol{\Xi}^{(0)}$ and the current iteration $k = 0$. The values for the stopping criteria, i.e., the required tolerance on $\Delta\boldsymbol{\Xi}^{\max}$ and the maximum number of iterations $k^{\max}$, can stem from a global setting.

Algorithm 1 Voltage parametrisation update

Require: $\boldsymbol{x}_0$, $\boldsymbol{t}$
Initialise: $\boldsymbol{\Xi}^{(0)}$, $\Delta\boldsymbol{\Xi}^{\max}$, $k^{\max}$, $k = 0$
while $\Delta\boldsymbol{\Xi}^{(k)} > \Delta\boldsymbol{\Xi}^{\max}$ and $k < k^{\max}$ **do**
  for component $i = 1, \ldots, m$ **do**
    Calculate current injections $\hat{\boldsymbol{i}}_i^C = h_i(\boldsymbol{t}, \boldsymbol{x}_{0,i}, \boldsymbol{\Xi}_i^{(k)})$ with a PINN
    Calculate Jacobian $\partial \hat{\boldsymbol{i}}_i^C / \partial \boldsymbol{\Xi}_i^{(k)}$
  end
  Calculate network injections $\hat{\boldsymbol{i}}^N(\boldsymbol{t}, \boldsymbol{\Xi}^{(k)})$
  Calculate network Jacobian $\partial \hat{\boldsymbol{i}}^N(\boldsymbol{t}, \boldsymbol{\Xi}^{(k)}) / \partial \boldsymbol{\Xi}^{(k)}$
  Calculate current balance error $\bar{\boldsymbol{\varepsilon}}$
  Calculate voltage parameter update $\Delta\boldsymbol{\Xi}^{(k)}$
  Update iteration $\boldsymbol{\Xi}^{(k+1)} = \boldsymbol{\Xi}^{(k)} + \Delta\boldsymbol{\Xi}^{(k)}$, $k = k + 1$
end
return $\boldsymbol{\Xi}^{(\text{final})}$

As we will describe in Section 7.7.2, we still observe a critical dependence of the accuracy on the time step size caused primarily by an inaccurate approximation of the bus voltages, i.e., when we choose a too small r. This limits the time step size Δt if we aim to remain within a specified

tolerance level. To simulate beyond this limit, we can apply the above procedure in a time-stepping fashion as laid out in Algorithm 2.

Algorithm 2 DAE simulator based on a time-stepping scheme

Require: $\boldsymbol{x}_0$, t_0, $t_{\max}$, Δt
Initialise: $\boldsymbol{x}'_0 = \boldsymbol{x}_0$, $t' = t_0$, $\boldsymbol{t}_{\text{eval}}$
1: **while** $t' \leq t_{\max}$ **do**
2: Define query points $\boldsymbol{t}$ on the interval $[t', t' + \Delta t]$
3: Calculate voltage parametrisation $\boldsymbol{\Xi}(\boldsymbol{t}, \boldsymbol{x}'_0)$
4: Evaluate and store trajectory $[\boldsymbol{t}_{\text{eval}}, \hat{\boldsymbol{x}}(\boldsymbol{t}_{\text{eval}}, \boldsymbol{x}'_0, \boldsymbol{\Xi})]$
5: Update initial values $\boldsymbol{x}'_0 \leftarrow \hat{\boldsymbol{x}}(\Delta t, \boldsymbol{x}'_0, \boldsymbol{\Xi})$ and $t' \leftarrow t' + \Delta t$
end
6: **return** Trajectory

With *PINNSim*, we refer to the combination of Algorithms 1 and 2 with PINNs used for the approximations of the state evolution $\hat{\boldsymbol{x}}$ and the current injection $\hat{\dot{\boldsymbol{i}}}^C$. PINNSim can also be run with Runge-Kutta (RK) schemes or differentiable ODE solvers as we describe in Sections 7.4 and 7.5 – only lines 3 and 4 in Algorithm 1 would be affected. However, the use of PINNs combines high approximation speed with high approximation accuracy. Importantly, this can hold even for large time step sizes; a RK-based or an ODE solver-based version of PINNSim will either be much slower (ODE solver-based) or limited to small time step size (RK-based) as we will show in the subsequent section. This shall highlight that the use of PINNs becomes the key to making PINNSim an attractive DAE solution algorithm. Needless to say, that there are many possible extensions to improve the efficiency of the algorithm such as an improved initialisation, adaptive time step sizes, and error estimation.

7.7 Experiments

This section shall demonstrate the characteristics of PINNSim. These results can be seen as a proof of concept and similar to Chapters 3 and 6 we focus mostly on describing the observed trends. We use the setup described in Section 7.7.1 to first illustrate the prediction for a single time step in Section 7.7.2 and then show results for a full trajectory in Section 7.7.3.

7.7.1 Setup

The 9-bus system serves as a test case for this section. We use the same setup that we described in Chapter 5 and applied in Chapter 6. We simulate a trajectory of up to 3 s with a change of the control input of generator 1 of $\eta = -0.5$.

The PINNs for the state approximations $\hat{\boldsymbol{x}}$ consist of 3 layers with 24 neurons each. We train these PINNs for a maximum state radius of $\|\boldsymbol{x}\|_{T,2} = 0.3$ (see definition in Section 3.3) and use a linear voltage parametrisation in the range of $\underline{\boldsymbol{\Xi}_i} = [\theta_{set} - \pi, V_{set} - 0.25, -3.5, -0.5]$ and $\overline{\boldsymbol{\Xi}_i} = [\theta_{set} + \pi, V_{set} + 0.25, 3.5, 0.5]$ dependent on the set-points θ_{set}, V_{set} from the steady-state power flow. For the present case, we extracted these values from the ground truth simulation of the system. In an application setting, these limits need to be chosen wide enough such that they cover all encountered states and voltage profiles.

For comparison, we also implement the following additional approximators, namely three explicit RK-based approximators (Forward-Euler (FE), Midpoint, 4th-order Runge-Kutta (RK4) – see

Section 2.2 for their definition), and an ODE solver. All of them need to be differentiable, which for the ODE solver is achievable using, e.g., torchdiffeq [138] or DifferentialEquations.jl [139]. While being the most accurate solution, the exact ODE solution requires a significantly longer solution time, therefore, it serves primarily as a reference value for the achievable accuracy. This reference value represents an "exact" solution given the voltage parametrisation, hence the error that we observe can be entirely attributed to the form of the voltage parametrisation and the formulation of its update. The explicit RK schemes on the other hand provide us with a benchmark for speed and illustrate the benefit that PINNs provide.

In addition, we also compare the results with an implementation of the trapezoidal rule. In fact, the trapezoidal rule is very similar to a scheme with a linear voltage profile $r = 1$ evaluated on two points, i.e., $s = 2$, at the beginning and the end of the time step. Only the state approximation is obtained with an implicit scheme instead of an explicit one.

For the error computations for a single time step, we use the final value of the prediction as they serve as the initial condition for the subsequent step. Its magnitude will be critical for a time-stepping-based simulator as errors will compound. For the analysis of an entire trajectory, we will use the maximum observed error. In case of the trapezoidal rule, we assume a linear evolution of all variables between t_0 and $t_0 + \Delta t$.

7.7.2 Single time step

We begin with tests on a single time step to obtain an understanding for the sources of approximation errors and the limitations that arise for the simulation of longer time periods, i.e., when applying a time-stepping scheme. As in previous chapters, we distinguish between questions of accuracy and speed.

Accuracy

There are two sources of errors in PINNSim. The first relates to the approximation of the voltage by the finite Taylor series. Based on this approximation, the components' state trajectories need to be evaluated which can incur further approximation error based on the used scheme. Crucially, both are dependent on the time step size Δt as the following presents. In a first analysis, we show in Figure 7.5 the maximum observed error for the difference between the rotor angle and voltage angle at the corresponding bus, i.e., $\max_i |\delta_i - \theta_i|$. Along the x-axis we increase the time steps size Δt and the y-axis corresponds to the use of different approximators for $\hat{\boldsymbol{x}}$. The different panels correspond to the order of the voltage approximation r. While we have implemented the RK and ODE solver-based state predictions up to order $r = 4$, the trapezoidal rule and the PINN-based approximators only appear in the first panel, i.e., for a linear voltage profile. For the trapezoidal rule, this is the case by construction. The PINN-based approximation could also be implemented for higher order voltage profiles. Such implementations will be addressed in future work. The colouring indicates the maximum absolute error at the end of the time step. We on purpose limited the colour scale to 1 deg. All tiles with a deep-yellow colour lead to errors larger than this threshold and should not be considered as accurate enough. When we apply the time-stepping algorithm even relatively small errors like 1 deg can quickly grow due to compounding effects. If many time steps are necessary, it is likely that this error tolerance needs to be significantly tighter to maintain high accuracy in the long-run.

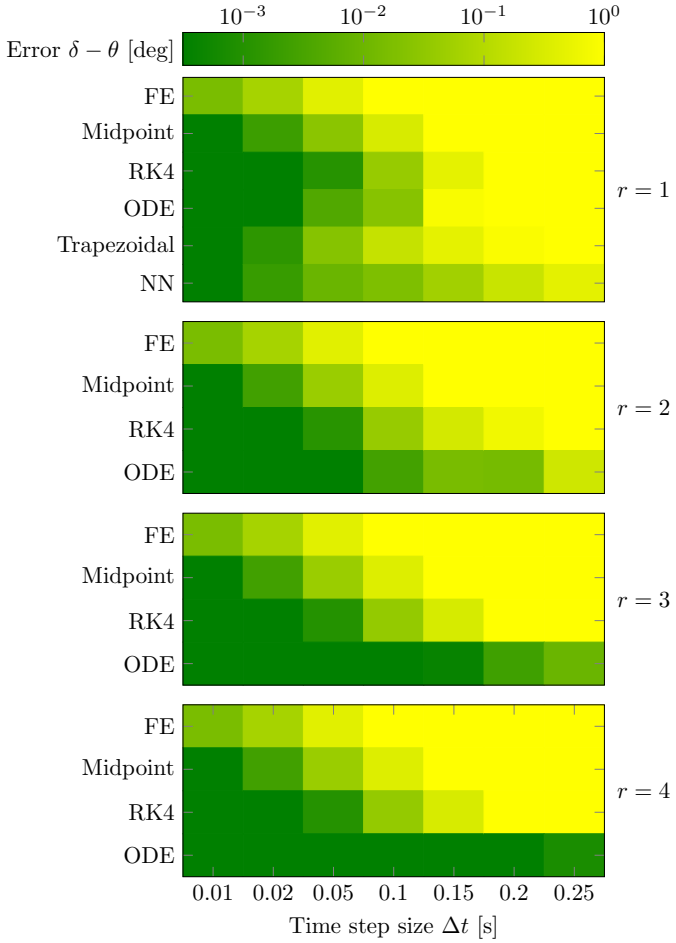


Figure 7.5: Maximum approximation error $|\delta_i - \theta_i|$ at the end of the time step as a function of the order of the voltage parametrisation r (rows), the predictor scheme (y-axis) and the time step size Δt (x-axis).

The main trend in Figure 7.5 is that higher orders of the voltage profile lead to significantly better predictions, given the use of an accurate approximation of the component dynamics. The ODE predictor can be seen as the most accurate predictor given a certain voltage profile. Out of the explicit RK schemes, only RK4 provides a viable alternative, as the FE and Midpoint predictor incur relatively high errors even for small time steps. As the errors do not reduce when increasing the voltage profile order, they can be attributed to the inaccuracy of the predictor scheme itself. Even for the RK4 scheme, we observe that the accuracy deteriorates for time steps larger than 0.1 s (in this setup). The trapezoidal rule performs very well for small time step sizes but reaches its accuracy limit also around 0.1 s; this originates from the local truncation error which scales as $\mathcal{O}\left((\Delta t)^{-3}\right)$. One observation sticks out though: The supposedly most accurate ODE solution performs not as well as expected for the linear voltage profile and worse than the PINN-based approach. The reason lies in the placement of the query points as the next paragraph explains. Overall, the PINN-based approximator yields the best results among the methods for $r = 1$. We show the corresponding results for the errors of the other variable types, i.e., $\Delta\omega$, θ, V, in Appendix B.2. The observed trends largely hold, for the parametrised variables θ and V, the predictor scheme has slightly less importance, instead, the order of the voltage profile gains importance.

The placement of the query points $\boldsymbol{t}$

To understand the better performance of the PINN-based approximator compared to the ODE solver-based approximator, we need to consider the placement of the query points $\boldsymbol{t}$. We are free to place these query points and choose their number, but these decisions will impact the update scheme of voltage parametrisation (see Algorithm 1). The following argument is close related to the one in Section 7.3.

The objective in the update scheme of the voltage parametrisation is the minimisation of a norm of the current error across the given time interval, as we formulated in (7.16)

$$\min_{\boldsymbol{\Xi}} \|\bar{\boldsymbol{\varepsilon}}(t, \boldsymbol{\Xi}; \boldsymbol{x}_0)\|, \quad t_0 \leq t \leq t_0 + \Delta t.$$

By evaluating the current error $\bar{\boldsymbol{\varepsilon}}$ on the selected query points $\boldsymbol{t}$, we approximate this objective value. As argued in Section 7.3, we can improve the accuracy of such an approximation by querying more points. However, for reasons of efficiency, we want to keep the number as low as possible without compromising too much on its accuracy. We illustrate what this means in Figure 7.6. In the

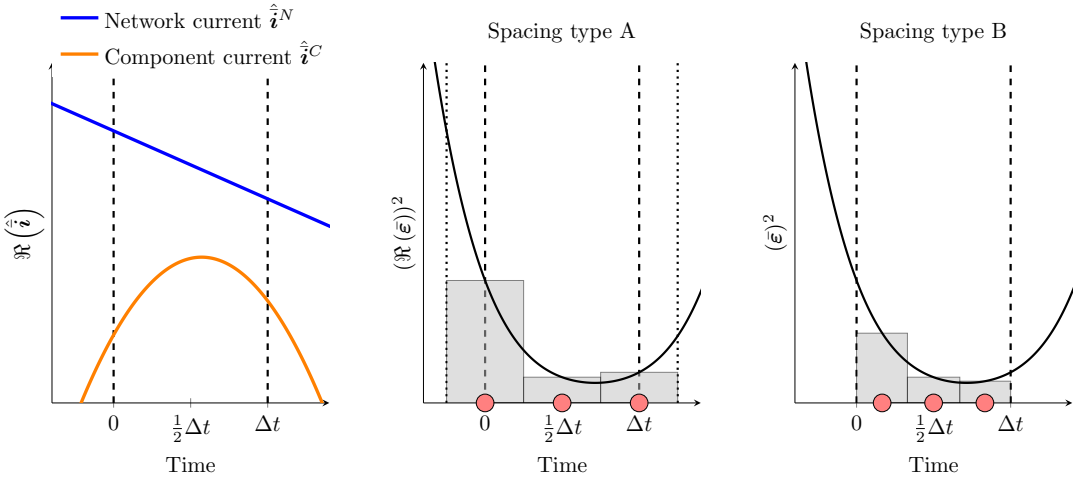


Figure 7.6: Interpretation of the query points as an integral approximation for the current difference between the network and component currents. For spacing type A, the covered area is wider than for spacing type B due to the placement of the query points (red marks).

left-most panel, we plot a network current approximation $\hat{\boldsymbol{i}}^N$ in blue and the corresponding current injection from a component $\hat{\boldsymbol{i}}^C$ in orange for an initial parametrisation $\boldsymbol{\Xi}^{(0)}$. By adjusting the voltage parametrisation $\boldsymbol{\Xi}$, we aim to reduce the current mismatch, i.e., the difference between the two curves, over the time step; it can be viewed as minimising an "area" between the two curves[6]. With the evaluation of the error at query points, we approximate this area in order to subsequently reduce its size. In the other two plots, we display the squared error on the y-axis which yields the black line for the given example. The two plots approximate the integral (the area under the curve) in two different ways. In the central plot, we place the query points at $\boldsymbol{t} = [0, 1/2, 1] \cdot \Delta t$, we call it "spacing type A". In the plot to the right, we choose $\boldsymbol{t} = [1/6, 3/6, 5/6] \cdot \Delta t$, named "spacing type B". The rectangles illustrate the resulting area approximations. The placement of the query points according to spacing type A includes the values at the boundary and therefore estimates the

[6]The exact interpretation of the term area depends on how the norm $\|\bar{\boldsymbol{\varepsilon}}\|$ is defined. We will use the L^2-norm, which is closely linked to the familiar Euclidean space and the common use of the term area.

area beyond the time step interval, indicated by the dotted lines. Hence, the error at the boundary becomes more influential. With spacing type B, we cover only the time interval. In the process of adjusting $\boldsymbol{\Xi}$, the black curve, i.e., the squared current error, will change and thereby also the area under the curve. The resulting values for $\boldsymbol{\Xi}$ hence depend on the covered area. This optimisation corresponds to finding the least-square fit that we use in (7.20).

The insight from this analysis is that spacing B is preferred as the resulting fit is optimised only for the time step and not beyond as it is the case for spacing type A. Hence, we applied spacing type B with $s = (r + 1) + 10$ whenever possible, i.e., for the PINN and RK-based approximators. For the ODE solver-based approximation, this attempt raised convergence difficulties as the initial condition of the algebraic variables can vary significantly. Hence, we use the spacing type A with $s = (r + 1) + 3$ as a compromise of accuracy and speed. For the linear voltage profile, i.e., $r = 1$, this limits the achievable accuracy, for higher order voltage profiles it becomes less of a concern as the overall current errors are much lower due to a better fit between network and component currents. This also means, that the ODE solver-based approximation becomes a good indication for the achievable accuracy and that well-trained PINNs with the spacing type B can be expected to yield similar accuracy.

Speed

In terms of speed, we have to consider not only the speed of the predictor itself as treated in Section 3.2, but also the iterative procedure to find the appropriate voltage parametrisation $\boldsymbol{\Xi}$. Hence, the number of iterations becomes an important factor, and we plot in Figure 7.7 the number of iterations in dependence of the time step size Δt, the predictor scheme, and the voltage profile order r. We observe that the number of iteration increases with higher order voltage profiles and increasing time step sizes. The exact number of iteration is also a matter of the set tolerance on the magnitude of $\Delta\boldsymbol{\Xi}$. Here, the termination criterion is set to $\Delta\boldsymbol{\Xi}^{\max} = 10^{-4}$ except for the trapezoidal rule ($\Delta\boldsymbol{\Xi}^{\max} = 10^{-8}$). The PINN-based approach can require many (up to the maximum of 20) iterations for large time steps. This is likely to be an effect of the insufficiently accurate voltage profile in combination with the prediction accuracy of the PINNs. Slight changes in $\boldsymbol{\Xi}$ can continue to alter the residuals but without converging to a local optimum and matching the termination criterion. We draw a similar conclusion for the RK4 predictor with $r = 3$ and $r = 4$. The higher number of iterations for large time step sizes compared to the ODE solver-based approximation indicates that inaccuracies in the state approximation $\hat{\boldsymbol{x}}$ will hamper the convergence of the voltage update scheme.

Comparing the actual run-times is only mildly insightful as none of the methods is optimised implementation-wise. On a per-iteration-basis, the trapezoidal scheme requires around 0.012 s, FE around 0.027 s, Midpoint around 0.033 s, RK4 around 0.042 s, and PINN around 0.021 s. These numbers will vary based on the order of the voltage profile and the number of query points. Overall, the trapezoidal rule delivers a powerful update scheme, that is fast and requires few iterations. However, accuracy requirements will limit the time step size which can offset these benefits of speed. Using a RK4 or PINN-based approximator might be more expensive per time step, but by using fewer time step, there is a potential speed-up.

By analysing the time-complexity of the different steps in the algorithm, we can obtain further insights. In the case of ideal parallelisation, the calculation of the current component injections and their Jacobians with respect to the voltage parameters $\boldsymbol{\Xi}$ can all be executed in parallel. The

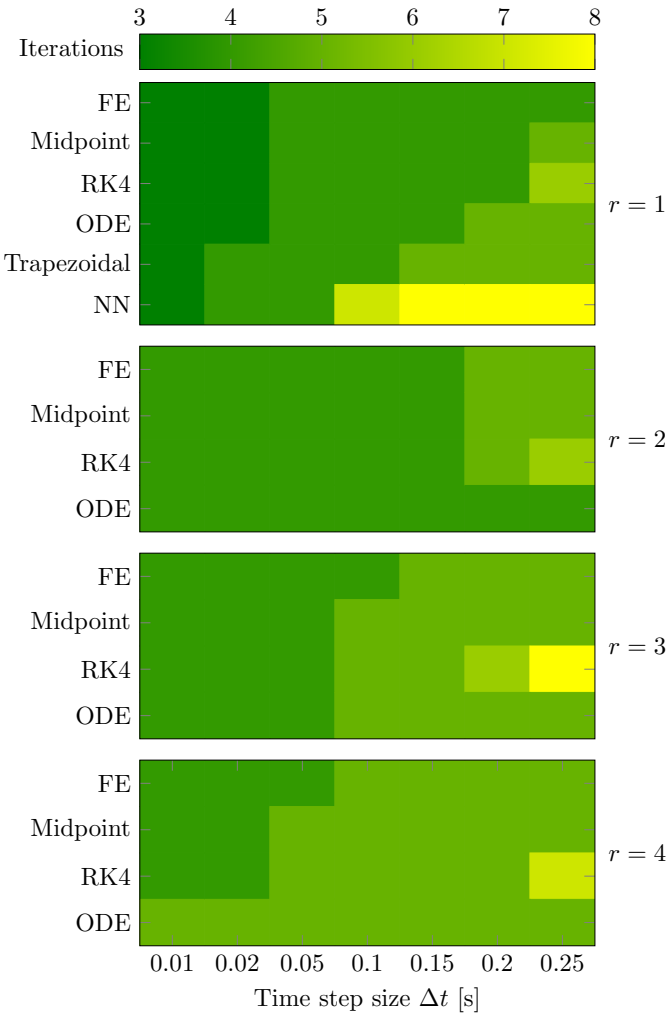


Figure 7.7: Number of iteration as a function of the order of the voltage parametrisation r, the predictor scheme and the time step size Δt.

run-time would then depend on the longest of these computations. By simultaneously evaluating the network currents and their Jacobian, all necessary computations to calculate the residual $\boldsymbol{\rho}$ and its Jacobian J can be performed in parallel. Hence, the run-time could be reduced in the ideal case to three times (1 forward and 2 backward passes) the run-time of the largest PINN or the run-time of evaluating the network equations (7.3a). The solution of the resulting least-square problem is then the other potentially time-consuming operation. It will scale with the number of buses n, the order of the voltage profile r and the number of query points s (we impose $s \geq r + 1$). In the worst case, the time-complexity of the least-square problem would scale as $\mathcal{O}(snr)^3$. However, thanks to the sparsity and known structure in the Jacobian, solving the least-square problem can be optimised and solved much faster. In the conducted experiments, it required only a small fraction of the overall computation time. However, for very large systems the solution of the least-square problem might become the main computational burden. Future work on PINNSim should therefore keep the potential for run-time optimisation in mind. The authors in [140] give an overview over many algorithms related to the optimisation of sparse linear systems that could be used as a starting point.

7.7.3 Multistep simulator

We now treat the prediction of an entire trajectory, i.e., applying the time-stepping simulator described in Algorithm 2. Given the analyses above, we use the RK4, ODE solver, and PINN-based approximators, the other predictors would require too small time steps for meaningful results. Apart from the trapezoidal rule, we use $s = (r+1)+3$ for all schemes, rendering the least-square problems over-determined. First, we analyse the resulting trajectories in the time domain to illustrate the simulator's characteristics, before presenting results to judge its overall accuracy and speed.

Trajectory analysis

We begin by presenting the results from the simulations in the time domain. In Figure 7.8, we display the resulting trajectories of the differential variables $\hat{\boldsymbol{x}}$ for a time step size of $\Delta t = 0.05\,\mathrm{s}$ in the left column and for a time step size of $\Delta t = 0.15\,\mathrm{s}$ in the right column. The used voltage profile is of order $r = 1$. For the smaller time step size, we observe overall very good agreement between

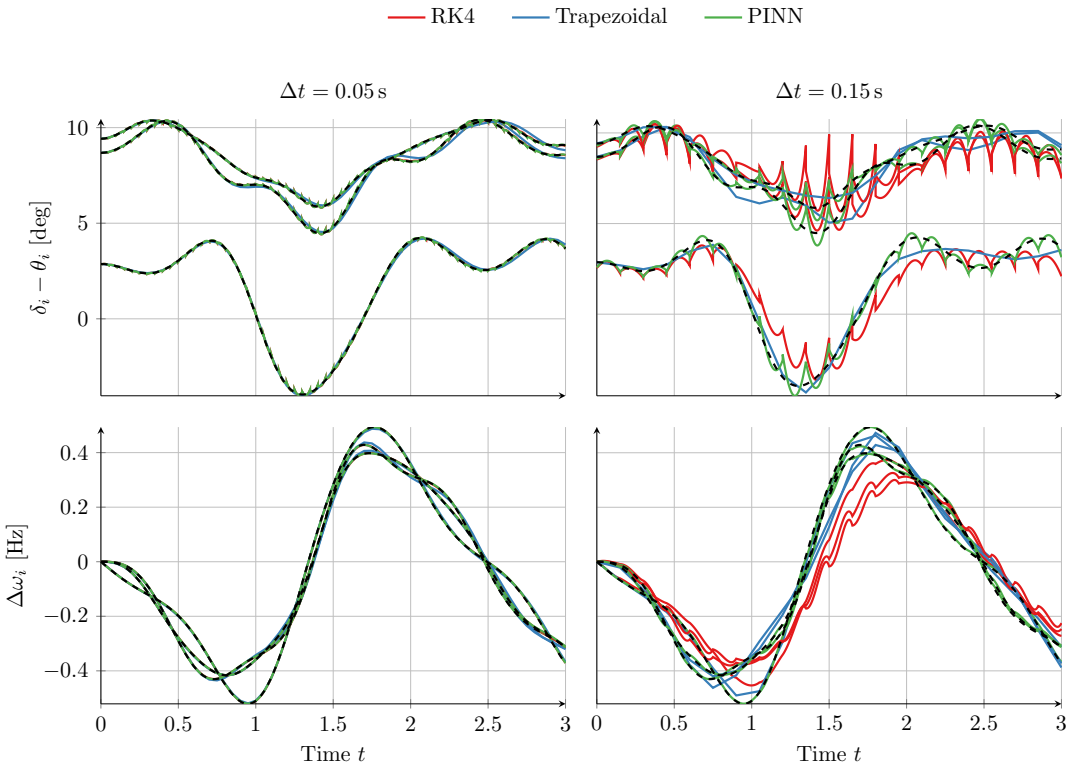


Figure 7.8: Prediction of the trajectory of the differential states for the three generators. The dashed black line represents the ground truth.

the ground truth (black dashed line) and the three prediction schemes. For the larger time step size in the right column, the RK4-based scheme (red) quickly deviates from the exact solution. The trapezoidal scheme (blue) remains closer to the exact solution, but also shows significant errors, best seen for the rotor angle. The PINN-based scheme approximates the frequency deviation $\Delta\omega$ very accurately and also follows the rotor angle trajectory well. However, the PINN and RK4-based predictions show a strange pattern for the rotor angle. The reason for this artefact is as follows: We plot the angle difference $\delta_i - \theta_i$. As the approximation of θ_i is restricted to a linear profile it cannot fit any curved trajectory. Because we placed the query points for the PINN and RK4-based schemes

according to the spacing type B (see the previous section), large errors can occur at the boundaries of the time step. This becomes even clearer when we consider the algebraic variables, i.e., the bus voltage approximations $\hat{\tilde{\boldsymbol{v}}}$, in Figure 7.9. For $\Delta t = 0.05\,\mathrm{s}$ all scheme resemble the ground truth

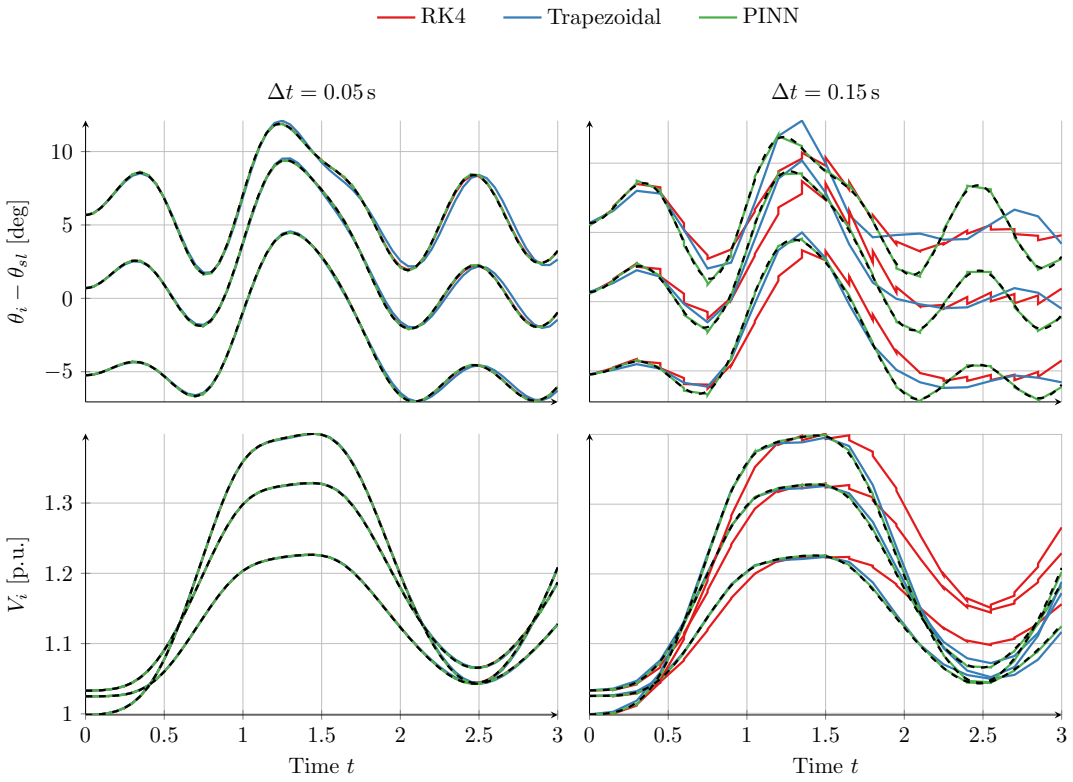


Figure 7.9: Prediction of the trajectory of the algebraic states at three buses. The dashed black line represents the ground truth.

closely, only the trapezoidal rule shows a slight deviation in the bus voltage angles. In contrast, the larger time step $\Delta t = 0.15\,\mathrm{s}$ leads to great inaccuracies for the trapezoidal and RK4-based scheme. Focussing on the voltage angles θ_i, we observe an instructive result. For the RK4-based scheme, the voltage angle shows jumps at the boundaries of the time steps. The trapezoidal scheme, in contrast, has no such jumps. By construction, the trapezoidal scheme will be consistent between time steps. Due to its discretisation for the update scheme, only the variables at the end of a time step are adjusted. The initial condition simply equals the value at the end of the previous time step. For the RK4 and PINN-based schemes we do not impose such a constraint. Instead, the update scheme relies on finding the fit with the lowest current error for the chosen query points. When the approximation of the states $\hat{\boldsymbol{x}}$ is accurate, it allows for a good trajectory approximation even for large time steps, as the PINN-based scheme shows. When the state approximation is too inaccurate, e.g., with a RK4 approximator, the resulting trajectories will also be inaccurate. The magnitude in the jumps of the voltage approximation $\hat{\tilde{\boldsymbol{v}}}$ can be an indication of insufficient approximation quality. This analysis can be understood even better when considering the current error $\bar{\boldsymbol{\varepsilon}}$. As the current balance (7.2a) needs to hold across for entire trajectory for an exact solution, the current error should, by intuition, also be low for an accurate approximation scheme. The small current errors for $\Delta t = 0.05\,\mathrm{s}$, shown in Figure 7.10, do align with this intuition. However, when we look at a larger time step size, i.e., $\Delta t = 0.15\,\mathrm{s}$, it becomes clear that the current error alone might not be sufficient for judging the solution accuracy. The RK4-based scheme shows current errors of similar

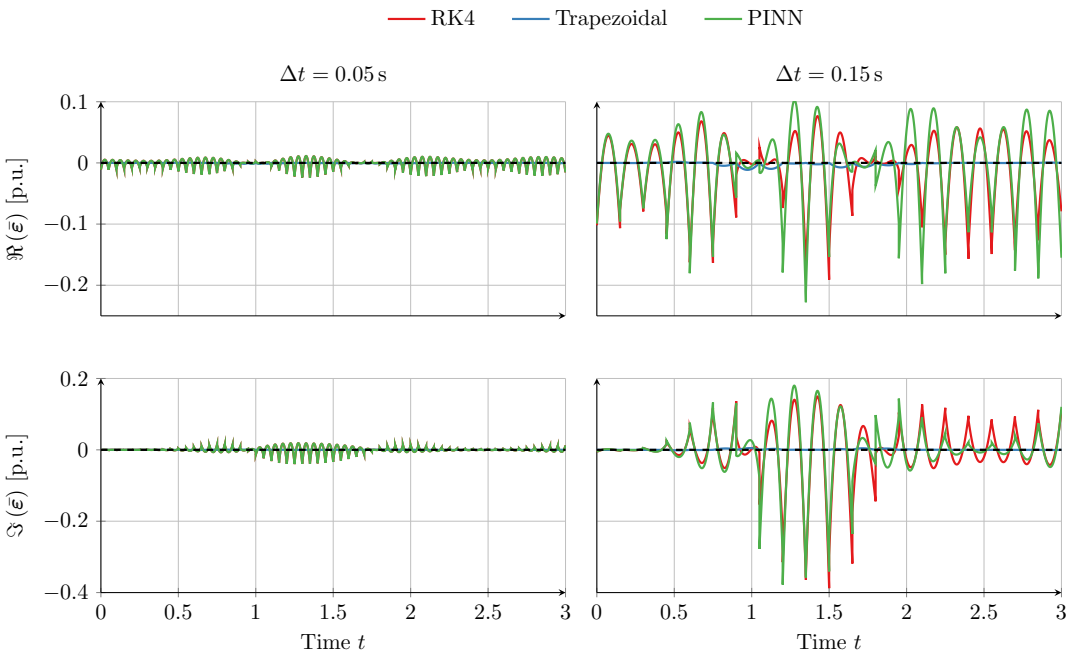


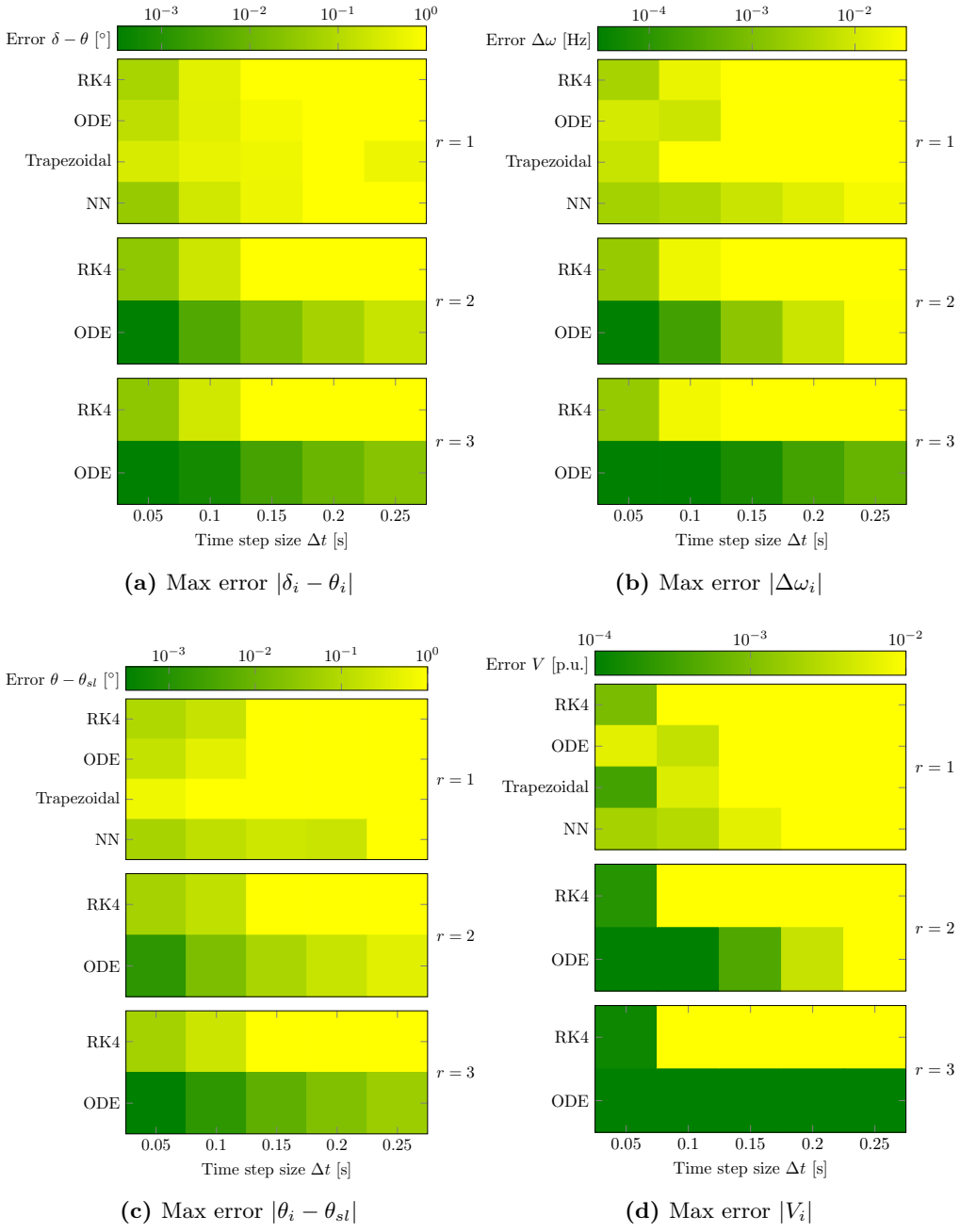
Figure 7.10: Current error $\bar{\epsilon}$ at a single bus.

magnitude as the PINN-based scheme. Still, the former is not accurate as seen in Figures 7.8 and 7.9. The reason is that we also require an accurate approximation of the state evolution (7.3c) given a voltage approximation $\hat{\bar{\boldsymbol{v}}}$. For a large time step size of $\Delta t = 0.15\,\mathrm{s}$, this requirement can be achieved with a PINN but not with a RK4 approximator, which shows that using PINNs plays a crucial role in PINNSim.

Accuracy and speed

Similar to the single time step analysis, we now plot the maximum errors of the differential and algebraic variables along the entire trajectory. Across all variables we see that the errors for the PINN-based simulator remain low, even for large time step sizes. When increasing the voltage profile order, the ODE-based simulator achieves accurate results for all tested time step size. Based on the previous results and arguments, PINNSim should be able to provide results of similar accuracy by training PINNs with higher-order voltage profiles.

To judge speed, we show in Figure 7.12 the average number of iterations per time step $\bar{k}$. The results align with the observations from the analysis of a single time step, which implies that the high number of iterations for the PINN-based simulator originate from the single time step behaviour. Hence, they should be addressed there to lower the number of iterations for each time step. The total simulation time then scales inversely proportional with the time step size and proportional to the cost per iteration C_I as discussed in Section 7.7.2. This leads to an overall time complexity of $\mathcal{O}(\bar{k}\, C_I\, (\Delta t)^{-1})$. The speed-improvements gained from larger time steps need to be weighted against potentially slower convergence of the voltage update scheme, i.e., larger $\bar{k}$.



(a) Max error $|\delta_i - \theta_i|$

(b) Max error $|\Delta\omega_i|$

(c) Max error $|\theta_i - \theta_{sl}|$

(d) Max error $|V_i|$

Figure 7.11: Maximum approximation error over the trajectory as a function of the order of the voltage parametrisation, the predictor scheme and the time step size.

7.8 Concluding remarks

This chapter has presented a proof-of-concept of a simulation algorithm for power system DAEs which we call *PINNSim*. It allows incorporating PINN-based flow approximators into a scalable and flexible solution algorithm. This could allow for very large time step sizes and thereby accelerate TDSs. The two key factors for ensuring the simulator's accuracy are the approximation quality of the algebraic variables, here the complex bus voltages $\hat{\tilde{\boldsymbol{v}}}$, and the state evolution $\hat{\boldsymbol{x}}$. The former can be achieved by an appropriate choice of the parametrisation of $\hat{\tilde{\boldsymbol{v}}}$, the latter hinges around well-trained PINNs. The second aspect highlights the need for the investigation carried out in

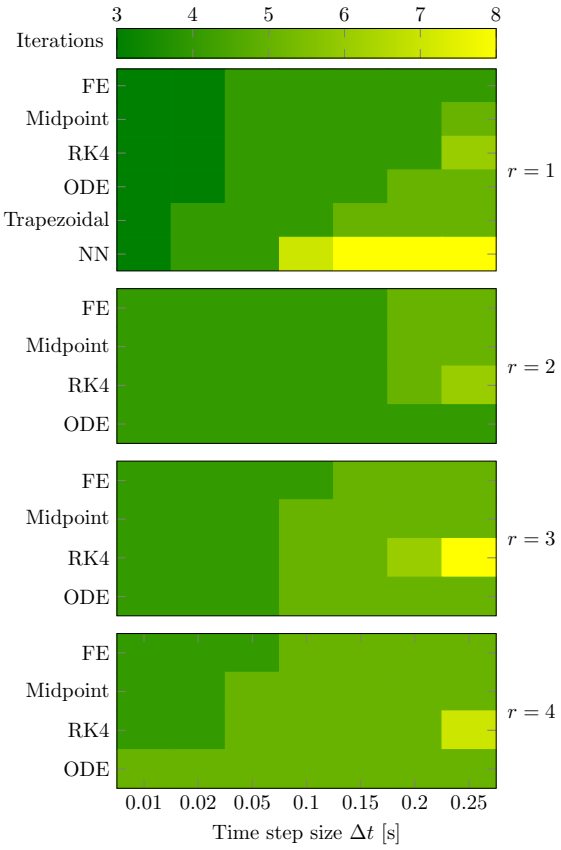


Figure 7.12: Average number of iterations across the simulation of the trajectory as a function of the order of the voltage parametrisation, the predictor scheme and the time step size.

Part I. A tool like PINNSim will require certain characteristics, e.g., consistent solutions. By defining a task $\mathcal{T}$ that represents these requirements, we can better align questions of how to train and how to use PINNs.

Future work could follow a few lines of ideas. First, finding appropriate voltage parametrisation schemes, i.e., basis functions, will be problem dependent. Polynomials are a straightforward choice and have proved their appropriateness for this case. However, trigonometric or exponential basis functions might offer the upside of using fewer parameters for more suitable approximations. Fewer parameters, i.e., smaller r, reduce the size of the parameter update scheme but also simplify the learning problem in terms of dimensionality.

A second line of work will concern the approximation of the component dynamics. Small time-scales and many states often complicate dynamic simulations. Including higher-order models and in particular models with smaller time-scales has therefore a high priority. The dynamics can be extended to include control schemes and potentially even discrete behaviour. In the latter, we might see non-smooth current injections, however, as long as the voltage profile can adjust accordingly, the algorithm should be able to accommodate such difficult characteristics. The use of PINNs could prove even more beneficial in such circumstances.

A third aspect is the optimisation of the voltage update scheme. It contains a lot of structure and ideas like dishonest Newton-Raphson schemes, restructuring the matrices and vectors for better

factorisation schemes, and improved initial parametrisation, e.g., based on the previous time step, might offer significant potential for speed-ups.

In general, the implementation of the entire algorithm in a compiled and potentially parallelisable fashion is necessary for assessing the realisable speed improvements. Given such an implementation, larger systems are then to be explored. Furthermore, a range of simulation scenarios, such as short circuits and line outages, needs to be test to demonstrate the robustness of the approach. Additionally, theoretical analyses on the error bounds and convergence will be important to place requirements on the PINN-based approximators. As argued in Chapter 3, given a task and performance criteria, it is more a question of how quick rather than if we can learn the desired approximators.

Part III

Conclusions on ML in power systems

CHAPTER 8

Trusting ML-based methods in power systems

In this chapter, we broaden our perspectives beyond the specifics of conducting Time-Domain Simulations (TDSs) with Physics-Informed Neural Networks (PINNs) and ask what Machine Learning (ML) models would need to provide to be embraced in power systems. To this end, we adopt the idea of viewing models along the two dimension of *usefulness* and *computational cost*, that we used to motivate this thesis in Chapter 1. If ML-based methods shall extend the region of applicable models, we, ultimately, need to confront the question: When do we trust ML-based methods in power system applications? In Section 8.1, we start by distinguishing two types of trust in a modelling context and then illustrate in Section 8.2 which problems ML models might be used for. In Section 8.3, we touch upon the point of trusting the process of learning models itself, i.e., increasing its robustness.

8.1 Trust in modelling and problem-solving

As the term *trust* is very hard to grasp and admits several nuances, we first always implicitly consider a specific problem or question for which we decide whether to trust ML or not; statements like "generally trusting or not trusting ML" cannot be derived here. Second, to make it a binary decision, we apply the rule that if one is willing to utilise the modelling approach in real world operation, i.e., one accepts the associated risk, one *trusts* the approach. Using a model may be out of conviction or necessity, as we will discuss later on, but the benefit must be greater than the risk associated with inaccuracies – this aligns with the idea of *usefulness* of models as described in Chapter 1. Broadly speaking, there is a distinction between the problem formulation and the solution method. As we argue in the following, trust in the former implies trust in the chosen representation of reality, whereas trust in the latter concerns trust in the obtained solution.

8.1.1 Trusting the problem formulation

As motivated in Chapter 1, the power system is an extremely complicated physical system. Describing it in all its detail leads to a model that might be useful for analysing many problems, however, it would exceed the computational budget by far, i.e., it is intractable. A basic and general optimisation problem could, for instance, be the provision of electricity at low cost while ensuring availability throughout 99.99% of the year. While we now might have a clear problem formulation, we cannot solve it. Out of the necessity to solve the problem of running the power grid, we formulate multiple sub-problems and many more sub-sub-problems, and so on, to obtain problem formulations that are actually tractable. Take the objective of cheap electricity: For short-term scheduling of generators, we could formulate an Optimal Power Flow (OPF) problem

considering a DC or an AC representation of the grid and include or exclude the uncertainty associated with renewable sources. The AC-OPF with uncertainty would be the most adequate to achieve (expected) cost optimality but if it is not tractable, we have to choose a simpler model. For a ten-year expansion planning even a full DC-representation of the grid might lead to intractability, hence, we might only use country-wise representations of the generating units. The choice of the used problem formulation needs to consider if a tractable solution algorithm exists.

This "chopping-up" or simplification of a big problem into many smaller sub-problems – essentially modelling – is the bread-and-butter of power system engineers; at an abstract level it concerns using strategies of Reduced-Order Modelling (ROM). While the process is a necessity to run the power system, it introduces sub-optimality and at times, even infeasibility between the solutions to the sub-problems. In many cases, the approach to obtain a ROM formulation is to simplify the physical model. Classic examples are the simplifications of the power grid from a non-linear AC-representation to a linear DC-representation to a copperplate model or the modelling of synchronous machines, starting from a flux-based model to the two-axis model to a classical machine model. This process requires balancing the loss of accuracy, which potentially compromises the model's usefulness, and the gain of tractability. The fact, that we may have undergone several steps of simplification beforehand, must always be kept in mind. The *modelling errors* in those steps might be much more severe than the one introduced for the current ROM step. A simple example is the question, whether a probabilistic DC-OPF is "better" than a deterministic AC-OPF? The answer is setting-specific and actually a matter of trust, in the sense of trusting the applicability of a problem formulation. As problem formulation defines a model, explicitly or implicitly, there will be a level of *usefulness* associated with it.

8.1.2 Trusting the solution algorithm

Whenever we settled for a problem formulation, we can ask whether the algorithm to solve the problem formulation, i.e., to evaluate the model, is trustworthy. Unless there exists an analytical solution, we will inevitably incur a numerical approximation error. This approximation error can reduce the usefulness of the model defined by the problem formulation. When we refer to a *solution*, we usually imply that the numerical approximation errors are negligible. However, from a perspective of usefulness, even a mildly inaccurate solution algorithm might not render its solution useless. In a way, a solution algorithm entails a penalty on the usefulness of the problem formulation. How big this *usefulness-penalty* is, depends on the errors that are made and their severity. Balancing this usefulness-penalty with the potential gains in terms of computational cost determines whether we will eventually trust the solution algorithm.

For the case of power system dynamics, one could consider if running a highly accurate TDS is actually of benefit or if a faster but less accurate simulation would suffice; the incurred usefulness-penalty will determine if the outcome is still considered useful. This decision becomes tightly linked to the problem formulation itself. If, for instance, a fast but slightly inaccurate solution algorithm allows the simulation of 1000 initial conditions instead of a single one, the information gained by those additional runs could outweigh their lower accuracy. The overall usefulness might improve because the faster algorithm enables us utilising more useful problem formulations (evaluating 1000 selected initial conditions instead of 1). The usefulness-penalty of the solution algorithm becomes less critical.

8.2 Which power system tasks to solve with ML?

With these considerations in mind, we attempt a classification of ML-based modelling approaches depending on the objective that is pursued. These objectives are formulated along the lines of altering the usefulness and the solution cost of models.

8.2.1 ML to solve forward problems: Accelerating model solution

The approach, that we explored in this thesis, is best described by accelerating the model solution. We solve the forward problem by learning an explicit function that is easy to evaluate. The modelling starts from a very clearly defined problem formulation which we already trust. In this thesis, the governing Ordinary Differential Equations (ODEs) and Differential Algebraic Equations (DAEs) constitute this problem formulation. When we turn this clearly defined setup into an optimisation problem for the ML algorithm, we aim to replicate a solution from another solution algorithm that has a negligible usefulness-penalty. By spending computational resources to learn the ML-based explicit approximator, that is fast to evaluate at run-time, we can accelerate solving a certain problem formulation with a small usefulness-penalty. As elaborated in Chapter 6, the curse of dimensionality will make approaches of this type, more and more difficult when the input dimensionality of the problem increases, unless we can induce problem specific knowledge, i.e., inductive biases. It is important to focus on reducing the usefulness-penalty in this context, not simply accuracy. If a solution trajectory becomes unstable, the identification of this qualitative behaviour might still be useful, even if the prediction of the numerical values itself is not entirely accurate. Such considerations need to be taken into account in the formulation of the learning task $\mathcal{T}$ and how to derive a loss function $\mathcal{L}$ from the performance metric $\mathcal{P}$. In fact, the performance metric should incentivise a low usefulness-penalty rather than just a high accuracy.

8.2.2 ML to solve inverse problems: Finding more useful models

Another task can be the identification of more useful models, i.e., solving inverse problems. Let us take the example of the dynamics of a complex component with many states, e.g., $\boldsymbol{x} \in \mathbb{R}^{10}$. When connected to the power grid, this leads to a time-varying current injection and its accurate approximation would be the aim when learning a forward problem. If we alter the task to instead find, for example, an alternative update function $\boldsymbol{f'} : \mathbb{R}^{2+1} \mapsto \mathbb{R}^2$ based on a two-dimensional state, i.e., $\boldsymbol{x'} \in \mathbb{R}^2$, we solve an inverse problem. Now, the objective becomes to find a model, in fact a problem formulation, at a reduced usefulness level, but that is faster to evaluate. By applying this strategy of ROM, one might even be able to make more useful problem formulations tractable. Evaluating 1000 reduced-order models can be more useful than one fully-detailed model. As reviewed in Section 4.5, the form of the update function can range from very specific model structures, e.g., an ODE to which we fit parameters, to very loose definitions of the update function as seen in Neural Ordinary Differential Equation (NeuralODE). The following difficulty arises with this type of ML-based modelling. The resulting models might not have concrete physical meaning as we are used to. It complicates the transformation between the reduced-order model and the detailed model, as the defining *embedding* is learned and not constructed using physics-based arguments. As a consequence, estimating the difference in the usefulness of the problem formulation using the detailed or reduced-order-model becomes more complicated. It heavily depends on how well the reduced-order-model generalises and whether there are variations across different operating points.

8.2.3 ML-based direct approaches: Forward and inverse problem united

The splitting of the main problem, i.e., running the power grid, into many small problems is heavily under the influence of either physically motivated simplifications, e.g., time-scale separation to reduce the dimensionality of dynamic components, or to produce formulations which can be efficiently solved. These guiding principles, physics-based or solution algorithm-focused, do not necessarily lead to attractive ML problem formulations. For example, solving a convex relaxation of an OPF problem is a possible learning task, but it might not be an attractive one when we could also learn the exact solution. The usefulness of the convex relaxation is linked to the solution algorithm that it enables and less to problem formulation itself. This shall say that ML might require from us to rethink the problem formulations themselves. TDSs, for instance, are an intermediate step for judging whether criteria for secure operation are met or not. Once we carried out the TDS, checking these criteria is simple. In principle, a ML-based model could perform the same task without going through the process of a running a TDS. The initial condition, control inputs, and system parameters would constitute the inputs to the ML model, which then directly predicts the classification of the system being secure or insecure. One might view it as solving an inverse and forward problem at the same time. The inverse problem concerns finding a useful representation for the stability assessment of the system based on the input data. The forward problem evaluates this representation and then performs the classification. We refer to such combined approach as *direct* approaches[1].

This leads us to a conflict of objectives: The resulting model is very powerful but not really interpretable. Constraining the hypothesis class h_θ to predict interpretable quantities on the other hand can inhibit the capacity of ML to abstract and find relations that are not easily derivable – if they were, there is a good chance we found them in more than 100 years of power systems research. The success of *deep learning*, as described in [49], relies on, to some extent, allowing this *uninterpretable* abstraction. In AlphaGo [142], for example, the approach to not explicitly specify how each move shall look enabled the algorithm to find moves that surprised even the best Go-players.

In practice, the curse of dimensionality again complicates such direct approaches. If we required large datasets based on simulations, we would quickly reach computational limits, if we used real-world measurement data, the question of generalisation immediately occurs. While we might have many data points to learn from, they are not necessarily favourably distributed. In particular, when rare events such as blackouts are concerned, the generalisation quality is questionable. Hence, these *direct* approaches offer large upside in terms of speed but need to consider how to handle the curse of dimensionality in a way that allows them to generalise well.

8.3 Trusting the ML-based modelling process

Let us return to the question on how to foster the applicability of ML-based methods for power systems. The distinction of the type of problem, we want to solve, plays a vital role. For pure forward methods, which received most attention in this thesis, we have very different metrics to consider than for an inverse problem. The perspectives given in Chapters 3 and 4 showed different

[1] In transient stability analysis, "direct" approaches such as those based on energy functions, see, e.g., [75], [141], can be seen as an equivalent. They avoid TDSs by defining an appropriate proxy (the energy function). Its generalisation capacity is rooted in the analytical derivation. The limitation of the method arises from where the derivation is not possible.

angles on "training a PINN"; they originated from the differences in the tasks, i.e., the type of problem. What they share though, throughout this thesis, is a large part of the process to solve them. This process includes potentially simulating a dataset, pre-processing it, training the model including a model selection, and assessing its performance. Controlling and improving this process in order to ensure reliable results for a broad range of tasks, can improve the overall trust in ML-based modelling by better aligning expectations with the characteristics of the resulting models.

For this purpose, the idea of *pipelines* or *frameworks* is omni-present in the data-driven modelling world, see, e.g., [143], [144]. Embracing the use of such frameworks can help the power system community to develop clarity on language, terminology, and standards. Ideally, this assists the "end-users" of ML models – in this thesis, for instance, grid operators – to easily determine the appropriateness of modelling assumptions and assess the learning outcome. To this end, the notion of a task $\mathcal{T}$ adopted from ML literature has been particularly helpful.

One easily overlooked aspect is that the software implementation benefits significantly from applying frameworks. It encourages task-independent implementations, hence they become easier to share and encourage standard interfaces between the steps in the ML process. The greater the ease of exchanging modules and testing alternative methods, e.g., another hypothesis class, the more we avoid underperforming models. On the software side, more and more platforms develop tools that make the construction of pipelines and handling models easier, e.g., [145]–[148]. Of course this must not lead to "blindly" applying ML methods – we saw the importance of understanding the underlying problem many times throughout this thesis – but it can help to screen promising approaches and build more trust in the ML-based modelling process.

CHAPTER 9
Conclusions and perspectives

9.1 Conclusions

This thesis aimed at investigating the modelling approach of Physics-Informed Neural Networks (PINNs) in the context of performing Time-Domain Simulations (TDSs) of power system dynamics. The benefit of PINNs is their fast evaluation speed paired with a sufficient solution accuracy. Eventually, however, the applicability of PINNs to power systems will hinge around their scalability to large multi-machine systems. Understanding if and how this scalability can be achieved was explored in this thesis.

To this end, we first provided a detailed formulation and analysis of the modelling approach that PINNs take. The clear alignment of PINNs with numerical approximation schemes for solving Ordinary Differential Equations (ODEs), namely Runge-Kutta (RK)-schemes, opens the door to an improved interpretation of what it means to learn the *flow* Φ of a dynamical system. It entails understanding error characteristics and taking advantage of the solution structure that numerical methods utilise. On this basis, we presented key factors for a successful modelling. We propose the use of a dimensionless state space that defines how the multi-objective optimisation of different state elements should be weighted. Furthermore, we introduce a loss-term, called dt-loss, that provides additional regularisation of the training. It penalises predictions that contradict the governing ODEs and it is evaluated on the supplied simulated data points. We introduce an extension to a model architecture that uses implicit-RK schemes. By including the time step size as an input variable, we transform the previously discrete-in-time approach to a continuous-in-time flow approximator. The task of identifying system parameters illustrated how imposing a regularisation based on the governing ODE affects the learning process.

In the second part of this thesis, we tested the scalability of PINNs in the context of TDSs for multi-machine systems. We find a great speed advantage over conventional methods at the cost of less accurate solutions. Still, the accuracy can be considered sufficient to assess the dynamic response, e.g., for screening tasks. The training, however, requires a considerable amount of time and does limit the ability to scale PINNs to realistic power system sizes, i.e., using them as a general-purpose solver for power system dynamics. To overcome this limitation we propose an algorithm called *PINNSim* that centres around approximating the dynamics of *single* machines (as investigated in the first part) with PINNs. The process of learning the dynamics of each component is performed separately from each other. To run a TDS with multiple machines, we formulate an optimisation problem that determines the interaction between the single machines. This optimisation problem is solved with an established root-finding algorithm. The potential acceleration originates in the possibility to perform larger and hence fewer time steps than with conventional methods. Thereby, PINNSim enables a scalable solution approach in which PINNs can be integrated in a modular fashion on a component-level.

Based on these insights, we conclude that the modelling framework of PINNs is best suited for dynamical systems with a relatively low dimensionality. In such settings, the learning remains tractable and building trust in the resulting models appears to be feasible. To achieve scalable simulation models with PINNs, hybrid approaches around decomposition techniques seem promising.

9.2 Future work

This thesis has provided a proof of concept of PINNSim that might offer the potential to use PINNs for realistic power system sizes and thereby accelerate TDSs. Our suggestions for future work are guided by the vision to develop PINNSim into a fully-fledged simulator that can be applied to a variety of power grids. Accuracy, efficiency, and reliability are three necessary objectives that we need to investigate in this context – on the level of single components (as in Chapter 3) and on the level of multi-machine systems (as in Chapter 7).

In the context of learning the dynamics (the flow) of a single component, we have to increase the complexity of the component dynamics to a level usually encountered in power systems. This can entail using components with more states, different time scales (stiff dynamics), or algebraic and discrete variables. If we also include various parameter settings and control inputs, it leads to a wide range of possible configurations. All these combinations constitute different but related tasks, hence profiting from already learned solutions should help to improve the efficiency and accuracy of the process, e.g., by applying transfer or multitask learning. By also standardising the process of learning the component dynamics, starting from the governing ODEs, the reliability of the results will improve – ideally all steps can be based solely on the definition of the component's ODEs. Detailed analyses of the error characteristics, maybe even the verification of the behaviour of PINNs, see, e.g., [149] for an overview, are further promising steps to improve the reliability and trustworthiness of this modelling approach.

When moving from single components back to the full simulator, accuracy, reliability, and efficiency are again three drivers of developing the approach. First and foremost, to achieve scalability, the efficiency of an iteration in the simulation must be improved. This primarily concerns the implementation of the voltage update scheme. Techniques like a dishonest Newton-Raphson or the exploitation of the sparsity in the calculations, are well-established in a similar context [4], [5], [19], hence, their applicability should be tested here as well. By analysing the occurring errors, adaptive time step sizes or ensuring tolerances might be possible. A crucial factor for the success of PINNSim is the choice of the representation of the voltage profile as it will determine the maximum allowable time step size. The intended use case as a general-purpose solver, furthermore, requires the investigation of the performance of PINNSim in different settings. Dynamics corresponding to set-point changes, topology changes or short-circuits are necessary capabilities. In light of the energy transition, the simulation of electromagnetic transients, so-called EMT-simulations, becomes increasingly important [7]. Based on this relevance and the potential for significant acceleration of the solution, we consider PINNSim applied to EMT simulations a promising future research direction.

APPENDIX A

Training and simulation setups

The following will describe the details of the training and simulation setup, i.e., parameters, software, and hyperparameters, used throughout the chapters.

A.1 Machine parameters

The parameters of the Single-Machine Infinite-Bus (SMIB) are shown in Table A.1.

Table A.1: SMIB parameters

Control input $\boldsymbol{u}$	Parameters $\boldsymbol{\lambda}$				
P [p.u.]	H [p.u.]	D [p.u.]	$E'_{q,0}$ [p.u.]	V [p.u.]	X'_d [p.u.]
0.1	2.0	2.5	1.0	1.0	2.5

For the simulation of the multi-machine systems, we use the settings shown in Tables A.2 to A.4. The control input values are chosen such that the set points for the power flow specified in [3], [132], [133] are met. We want to note, that our goal for the selection of parameters was not to model a *realistic* setup. Instead, the focus lay on generating a dynamic behaviour on which we could test the approaches in Chapters 6 and 7.

Table A.2: Machine parameters 9-bus system

Generator ID	Control input $\boldsymbol{u}$		Parameters $\boldsymbol{\lambda}$		
	P_m [p.u.]	E_{fd} [p.u.]	H [p.u.]	D [p.u.]	X'_d [p.u.]
1	0.9031	1.0560	23.64	2.364	0.0608
2	1.3428	1.0398	6.4	1.28	0.1198
3	0.9423	1.0184	3.01	0.903	0.1813

A.2 Software implementation and Hardware

All simulations and Neural Network (NN) training procedures are implemented in Python 3.9. The simulations utilise *Assimulo* [131], whenever we require the solution of Differential Algebraic Equations (DAEs). For Ordinary Differential Equations (ODEs), we use the package *torchdiffeq* [138] due to the possibility for differentiating through the solver. It is based on PyTorch [68], which we also use for all training procedures. The only exception forms the training of the Bayesian Neural Networks (BNNs) and Bayesian Physics-Informed Neural Networks (BPINNs) in Chapter 4.

Table A.3: Machine parameters 11-bus system

Generator ID	Control input $\boldsymbol{u}$		Parameters $\boldsymbol{\lambda}$		
	P_m [p.u.]	E_{fd} [p.u.]	H [p.u.]	D [p.u.]	X'_d [p.u.]
1	7.0000	1.1132	58.500	10.0	0.033
2	7.0000	1.1117	58.500	10.0	0.033
3	7.1909	1.1116	55.575	10.0	0.033
4	7.0000	1.1012	55.575	10.0	0.033

Table A.4: Machine parameters 39-bus system

Generator ID	Control input $\boldsymbol{u}$		Parameters $\boldsymbol{\lambda}$		
	P_m [p.u.]	E_{fd} [p.u.]	H [p.u.]	D [p.u.]	X'_d [p.u.]
1	6.7159	1.1586	500.0	100.0	0.05
2	6.4600	1.2231	30.3	6.06	0.05
3	6.7116	1.2272	35.8	7.16	0.05
4	6.5200	1.1125	38.6	7.72	0.05
5	5.0800	1.1108	26.0	5.20	0.05
6	6.6145	1.2062	34.8	6.96	0.05
7	5.8000	1.1225	26.4	5.28	0.05
8	5.6400	1.0713	24.3	4.86	0.05
9	6.5403	1.0563	34.5	6.90	0.05
10	6.8959	1.1275	42.0	8.40	0.05

In said chapter, we use *NumPyro* [107] and *Pyro* [108], that are specifically designed for probabilistic modelling, alongside *JAX* [70]. To track the high number of experiments in this thesis, we utilised *Wandb* [146] for logging, saving models, and performing hyperparameter tuning.

The used hardware is an AMD Ryzen 7 PRO (1.9 GHz, 8 cores) and 16GB RAM for the simulations and NN training in Chapters 4 and 7. For the training of the models in Chapters 3 and 6, we utilise a High-Perfomance Computing (HPC) cluster located at the Technical University of Denmark [150]. The nodes comprise 2xIntel Xeon Processor 2650v4 (12 core, 2.20GHz) and 256 GB memory of which we used 4 cores per training run.

A.3 Hyperparameters

A.3.1 Chapter 3

For the different setups, we ranked the model complexity for different combinations of the number of hidden layers N_L and the neurons per layer N_N. A model complexity of 1 corresponds to a *low* model complexity and 15 to a *high* model complexity. The ranking is based on the size of the training error when not stopping the training early, i.e., we test how well the training dataset can be fit. The order was very consistent over multiple setups.

The dataset and sampling strategy form additional hyperparameters. For the former, we explicitly state them throughout the chapter, for the latter, the linear sampling in time and annular sampling

Table A.5: NN model complexity in Chapter 3

Model complexity	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N_L	1	1	1	1	1	2	2	3	2	3	2	3	2	3	3
N_N	16	24	32	48	64	16	24	16	32	24	48	32	64	48	64

of the initial condition are chosen. We subtract 20% of the training dataset size to form the validation dataset. To avoid rerunning the simulation process needed for the dataset generation, we simulate 10000 data points once and then draw random samples out of this large dataset to form the training / validation dataset. For the test dataset, we simulated additional 2000 points.

The hyperparameter settings are not altered within the chapter, i.e., all models were run with the setup of the Limited Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) optimiser as implemented in PyTorch (`https://pytorch.org/docs/stable/generated/torch.optim.LBFGS.html`) in Table A.6. The loss weighting factors α_{dt}, $\alpha_{c,0}$, $\alpha_{c,\max}$ and the fade-in speed E' only assume the following values when dtNNs or Physics-Informed Neural Networks (PINNs) were trained, see Section 3.5.1. We tested a range of random hyperparameters for models of medium complexity, but no clear trends in the performance were observable. Trends were observed in terms of the training time, i.e., the number of iterations and the history size affect the speed per epoch. As the training can yield very low loss values, we internally scaled the loss in each epoch with the epoch count E. Thereby, the tolerance settings of the optimiser were not limiting. If they are active, the training will stop, potentially prematurely. This can lead to an undesired distortion of the accuracy if the model could have fitted the data better. The scaling of $\mathcal{L}$ with E avoided this phenomenon.

Table A.6: Hyperparameter settings Chapter 3

Learning rate	Tolerance change	Tolerance grad	History size	Line search	Max iterations	α_{dt}	$\alpha_{c,0}$	$\alpha_{c,\max}$	E'
1.0	10^{-9}	10^{-9}	120	False	25	0.1	10^{-6}	0.1	10

A.3.2 Chapter 4

In this chapter, we used for all setups, i.e., of NNs, BNNs, PINNs, BPINNs, the same model size / complexity, namely with $N_L = 2$ and $N_N = 16$. As we knew the ground truth data and aimed for an illustration of the differences between PINNs and BPINNs, the hyperparameters were not optimised and chosen with the knowledge of the exact parameters in mind, see Table A.9. For a parameter estimation without knowing ground truth a hyperparameter selection routine has to be implemented, e.g., based on a validation dataset.

For the BNNs and BPINNs, we use the No-U-turn-sampler [106] as implemented in NumPyro [107] (`https://num.pyro.ai/en/stable/mcmc.html#hmc`). We use an adaptable mass matrix, adaptable time step sizes, and a tree depth of 8 in the experiments. We sample 5000 warm-up samples and another 5000 samples for the estimation.

Table A.7: Hyperparameter settings for NNs / PINNs Chapter 4

Learning rate	Tolerance change	Tolerance grad	History size	Line search	Max iterations	α_c
0.1	10^{-10}	10^{-10}	100	False	25	1.0

A.3.3 Chapter 6

For the scaling of the states $\boldsymbol{x}$ and $\boldsymbol{y}$, we adopted the following strategy. We group the variables in four *variable groups* with the goal to find one common scaling for each group. The four groups are constructed by distinguishing between the relative rotor angles $\delta_i - \theta_i$, the frequency deviations $\Delta\omega_i$, the relative bus voltage angles $\theta_i - \theta_{sl}$, and the voltage magnitudes. We calculate the standard deviation across the values of all data points within each group. This forms the scaling factor ξ_i by which the loss in the transformed state space is calculated. This choice should also be regarded a hyperparameter as it will influence how well the models generalise for the different states.

We proceed analogous to the description above on Chapter 3. First, we define in Table A.8 the levels of model complexity (the values differ from Table A.5). The dataset sizes are as specified in Section 6.1, for the 11-bus and 39-bus systems, we used scenario E. The hyperparameters are selected as shown in Table A.9, the dtNN and PINN specific setting were only use for the 9-bus system in Section 6.3 for the training-time analysis.

Table A.8: NN model complexity Chapter 6

Model complexity	1	2	3	4	5	6	7	8	9	10	11	12
N_L	1	1	1	1	2	2	2	2	3	3	3	3
N_N	16	32	64	128	16	32	64	128	16	32	64	128

Table A.9: Hyperparameter settings Chapter 6

Learning rate	Tolerance change	Tolerance grad	History size	Line search	Max iterations	α_{dt}	$\alpha_{c,0}$	$\alpha_{c,\max}$	E'
1.0	10^{-10}	10^{-10}	80	False	18	0.1	10^{-10}	0.1	10

A.3.4 Chapter 7

For the three machines, we follow the setup from Chapter 3. All models are trained as PINNs with $|\mathcal{D}_{\text{Train}}| = 2000$ data points and $|\mathcal{D}_c| = 1000$ collocation points.

APPENDIX B

Additional results

B.1 Chapter 6

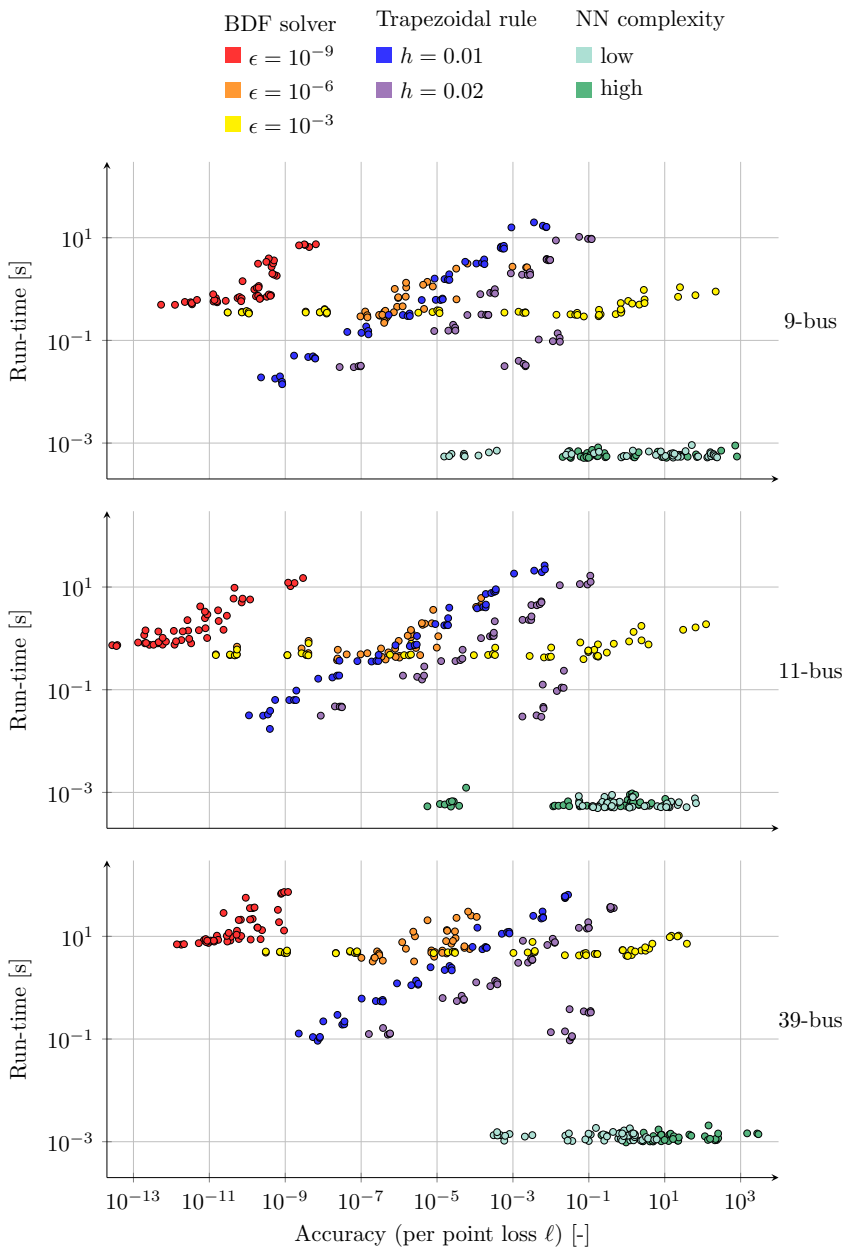
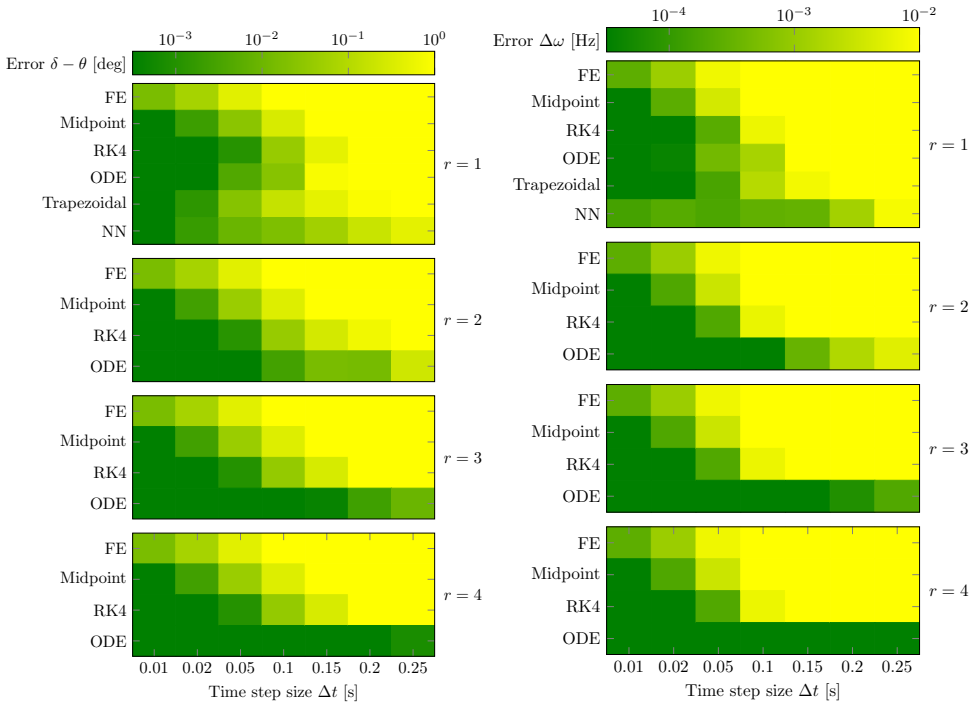


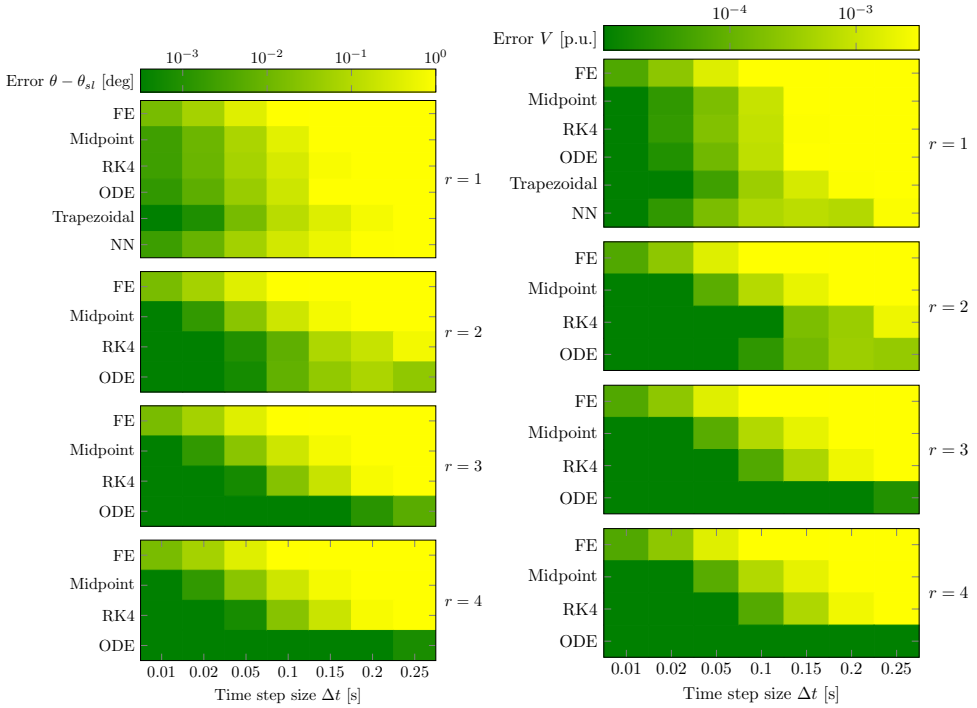
Figure B.1: Trade-off between run-time and accuracy for the 9, 11, and 39-bus system on a per-point basis. The point stem from 5 values of η and the time steps as in Figure 6.2, i.e., between 0.01 s and 5 s.

B.2 Chapter 7



(a) Max error $|\delta_i - \theta_i|$

(b) Max error $|\Delta\omega_i|$



(c) Max error $|\theta_i - \theta_{sl}|$

(d) Max error $|V_i|$

Figure B.2: Maximum approximation error at the end of a time step Δt as a function of the order of the voltage parametrisation (rows), the predictor scheme (y-axis) and the time step size (x-axis).

APPENDIX C

Inverse PINNs

C.1 MLE and MAPE of the trajectory

The approximation of a credible $\hat{\boldsymbol{y}}$ by a h_θ

$$\hat{\boldsymbol{y}} = \boldsymbol{g}(h_\theta(t)) \tag{C.1}$$

parametrised by $\boldsymbol{\theta}$ will be the first objective, so we formulate the likelihood $p(\mathcal{D}|\boldsymbol{\theta}, \sigma_y)$

$$p(\mathcal{D}|\boldsymbol{\theta}, \sigma_y) = \prod_{i=1}^{|\mathcal{D}|} \frac{1}{\sqrt{2\pi {\sigma_y}^2}} \exp\left(-\frac{\left\| \hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)} \right\|_{T,2}^2}{2{\sigma_y}^2} \right). \tag{C.2}$$

To obtain the Maximum Likelihood Estimator (MLE) $\hat{\boldsymbol{\theta}}_{MLE}$, we use (4.3) and apply the natural logarithm

$$\hat{\boldsymbol{\theta}}_{MLE}, \hat{\sigma}_{y,MLE} = \arg\max_{\boldsymbol{\theta},\sigma_y} |\mathcal{D}| \log\left(\frac{1}{\sqrt{2\pi\sigma_y^2}} \right) + \sum_{i=1}^{|\mathcal{D}|} \left(-\frac{\left\| \hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)} \right\|_{T,2}^2}{2{\sigma_y}^2} \right) \tag{C.3}$$

$$= \arg\min_{\boldsymbol{\theta},\sigma_y} \frac{|\mathcal{D}|}{2} \log\left(2\pi\sigma_y^2\right) + \frac{1}{2{\sigma_y}^2} \sum_{i=1}^{|\mathcal{D}|} \left\| \hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)} \right\|_{T,2}^2 \tag{C.4}$$

$$= \arg\min_{\sigma_y} \frac{1}{2} \log\left(2\pi\sigma_y^2\right) + \frac{1}{2{\sigma_y}^2} \left(\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left\| \hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)} \right\|_{T,2}^2 \right) \tag{C.5}$$

$$= \arg\min_{\sigma_y} \frac{1}{2} \log\left(2\pi\sigma_y^2\right) + \frac{1}{2{\sigma_y}^2} \left(\min_{\boldsymbol{\theta}} \mathcal{L}_y \right) \tag{C.6}$$

The inner minimisation problem corresponds to the usual loss as in (3.7a) (there it is $\mathcal{L}_x$ as we use $\boldsymbol{x}$ instead of $\boldsymbol{y}$). The MLE of σ_y can then be calculated by setting the derivative of the objective value to 0

$$0 = \frac{1}{4\pi {\sigma_y}^2} 4\pi\sigma_y - \frac{1}{{\sigma_y}^3} \mathcal{L}_y \tag{C.7}$$

$$0 = \frac{1}{\sigma_y} \left(1 - \frac{1}{{\sigma_y}^2} \mathcal{L}_y \right) \tag{C.8}$$

$$\hat{\sigma}_{y,MLE} = \sqrt{\mathcal{L}_y} \tag{C.9}$$

In case of the Maximum a Posteriori Estimator (MAPE), we need to include priors on $\boldsymbol{\theta}$ and σ_y, i.e., $p(\boldsymbol{\theta}), p(\sigma_y)$, by multiplying the likelihood with it. For the case of $\boldsymbol{\theta}$ we assume a multivariate Gaussian prior $\mathcal{N}(\mathbf{0}, I)$ and for σ_y we use a log-normal prior as we are likely to know the order of magnitude of the noise but not the exact size.

$$p(\boldsymbol{\theta}) = (2\pi)^{-|\boldsymbol{\theta}|/2} \det(I)^{-1/2} \exp\left(-\frac{1}{2} \|\boldsymbol{\theta}\|_2^2 \right) \tag{C.10}$$

$$p(\sigma_y) = \frac{1}{\sigma_y \sqrt{2\pi\sigma_\sigma^2}} \exp\left(-\frac{(\ln(\sigma_y - \mu_\sigma))^2}{2\sigma_\sigma^2} \right). \tag{C.11}$$

When adding these terms to the MLE after applying the logarithm, they become the following summands in the maximisation problem or with the opposite sign in the minimisation problem

$$\log(p(\boldsymbol{\theta})) = -\frac{1}{2}\left|\boldsymbol{\theta}\right|\log(2\pi) - \frac{1}{2}\left\|\boldsymbol{\theta}\right\|_2^2 \tag{C.12}$$

$$\log(p(\sigma_y)) = -\log(\sigma_y) - \frac{1}{2}\log(2\pi\sigma_\sigma^2) - \frac{1}{2\sigma_\sigma^2}(\log(\sigma_y - \mu_\sigma))^2. \tag{C.13}$$

The MAPE becomes

$$\begin{aligned}\hat{\boldsymbol{\theta}}_{MAPE}, \hat{\sigma}_{y,MAPE} = \arg\min_{\boldsymbol{\theta},\sigma_y} &\frac{|\mathcal{D}|}{2}\log\left(2\pi\sigma_y^2\right) + \frac{1}{2{\sigma_y}^2}\sum_{i=1}^{|\mathcal{D}|}\left\|\hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)}\right\|_{T,2}^2 \\ &+ \frac{1}{2}\left|\boldsymbol{\theta}\right|\log(2\pi) + \frac{1}{2}\left\|\boldsymbol{\theta}\right\|_2^2 \\ &+ \log(\sigma_y) + \frac{1}{2}\log(2\pi\sigma_\sigma^2) + \frac{1}{2\sigma_\sigma^2}(\log(\sigma_y - \mu_\sigma))^2.\end{aligned} \tag{C.14}$$

$$\begin{aligned}= \arg\min_{\boldsymbol{\theta},\sigma_y} &\log\left(2\pi\sigma_y^2\right) + \frac{1}{{\sigma_y}^2}\frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left\|\hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)}\right\|_{T,2}^2 \\ &+ \frac{1}{|\mathcal{D}|}\left|\boldsymbol{\theta}\right|\log(2\pi) + \frac{1}{|\mathcal{D}|}\left\|\boldsymbol{\theta}\right\|_2^2 \\ &+ \frac{2}{|\mathcal{D}|}\log(\sigma_y) + \frac{1}{|\mathcal{D}|}\log(2\pi\sigma_\sigma^2) + \frac{1}{\sigma_\sigma^2}\frac{1}{|\mathcal{D}|}(\log(\sigma_y - \mu_\sigma))^2.\end{aligned} \tag{C.15}$$

$$\begin{aligned}= \arg\min_{\boldsymbol{\theta},\sigma_y} &\log\left(2\pi\sigma_y^2\right) + \frac{1}{|\mathcal{D}|}\left|\boldsymbol{\theta}\right|\log(2\pi) + \frac{2}{|\mathcal{D}|}\log(\sigma_y) \\ &+ \frac{1}{|\mathcal{D}|}\log(2\pi\sigma_\sigma^2) + \frac{1}{\sigma_\sigma^2}\frac{1}{|\mathcal{D}|}(\log(\sigma_y - \mu_\sigma))^2 \\ &+ \frac{1}{{\sigma_y}^2}\left(\frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left\|\hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)}\right\|_{T,2}^2 + \frac{{\sigma_y}^2}{|\mathcal{D}|}\left\|\boldsymbol{\theta}\right\|_2^2\right)\end{aligned} \tag{C.16}$$

By fixing σ_y in the last row / bracket in (C.16), its value can be optimised for $\boldsymbol{\theta}$ separately, and it corresponds to a L^2-regularised training.

C.2 MLE of the ODE parameters

We can follow the same procedure as above to derive the MLE for the ODE parameters $\boldsymbol{\lambda}$ based on the likelihood functions (4.12) and (4.15).

$$p(\mathcal{D}, \mathcal{D}_c|\boldsymbol{\theta}, \sigma_y, \boldsymbol{\lambda}, \sigma_f) = p(\mathcal{D}|\boldsymbol{\theta}, \sigma_y)\; p(\mathcal{D}_c|\boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma_f) \tag{C.17}$$

$$\hat{\boldsymbol{\theta}}_{MLE}, \hat{\sigma}_{y,MLE}, \hat{\boldsymbol{\lambda}}_{MLE}, \hat{\sigma}_{f,MLE} = \arg\max_{\boldsymbol{\theta},\sigma_y,\boldsymbol{\lambda},\sigma_f} \log\left(p(\mathcal{D}, \mathcal{D}_c|\boldsymbol{\theta}, \sigma_y, \boldsymbol{\lambda}, \sigma_f)\right) \tag{C.18}$$

$$= \arg\max_{\boldsymbol{\theta},\sigma_y,\boldsymbol{\lambda},\sigma_f} \log\left(p(\mathcal{D}|\boldsymbol{\theta}, \sigma_y)\right) + \log\left(p(\mathcal{D}_c|\boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma_f)\right) \tag{C.19}$$

$$\begin{aligned} = \arg\min_{\boldsymbol{\theta},\sigma_y,\theta,\sigma_f} & \frac{|\mathcal{D}|}{2}\log\left(2\pi\sigma_y^2\right) + \frac{1}{2{\sigma_y}^2}\sum_{i=1}^{|\mathcal{D}|}\left\|\hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)}\right\|_{T,2}^2 \\ & + \frac{|\mathcal{D}_c|}{2}\log\left(2\pi\sigma_f^2\right) + \frac{1}{2\sigma_f^2}\sum_{i=1}^{|\mathcal{D}_c|}\left\|\hat{\boldsymbol{f}}(t^{(i)}, \boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma_f) - \mathbf{0}\right\|_{T,2}^2 \end{aligned} \tag{C.20}$$

$$\begin{aligned} = \arg\min_{\boldsymbol{\theta},\sigma_y,\theta,\sigma_f} & \log\left(2\pi\sigma_y^2\right) + \frac{|\mathcal{D}_c|}{|\mathcal{D}|}\log\left(2\pi\sigma_f^2\right) \\ & + \frac{1}{{\sigma_y}^2}\frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left\|\hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)}\right\|_{T,2}^2 \\ & + \frac{1}{\sigma_y^2}\frac{\sigma_y^2}{\sigma_f^2}\frac{|\mathcal{D}_c|}{|\mathcal{D}|}\frac{1}{|\mathcal{D}_c|}\sum_{i=1}^{|\mathcal{D}_c|}\left\|\hat{\boldsymbol{f}}(t^{(i)}, \boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma_f) - \mathbf{0}\right\|_{T,2}^2 \end{aligned} \tag{C.21}$$

By denoting the familiar loss terms by

$$\mathcal{L}_y = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left\|\hat{\boldsymbol{y}}(t^{(i)}) - \boldsymbol{y}^{(i)}\right\|_{T,2}^2 \tag{C.22}$$

and

$$\mathcal{L}_c = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}_c|}\left\|\hat{\boldsymbol{f}}(t^{(i)}, \boldsymbol{\theta}, \boldsymbol{\lambda}, \sigma_f) - \mathbf{0}\right\|_{T,2}^2 \tag{C.23}$$

we obtain the expression

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{MLE}, \hat{\sigma}_{y,MLE}, \hat{\boldsymbol{\lambda}}_{MLE}, \hat{\sigma}_{f,MLE} = \arg\min_{\boldsymbol{\theta},\sigma_y,\boldsymbol{\lambda},\sigma_f} & \log\left(2\pi\sigma_y^2\right) + \frac{|\mathcal{D}_c|}{|\mathcal{D}|}\log\left(2\pi\sigma_f^2\right) \\ & + \frac{1}{\sigma_y^2}\left(\mathcal{L}_y + \frac{\sigma_y^2}{\sigma_f^2}\frac{|\mathcal{D}_c|}{|\mathcal{D}|}\mathcal{L}_c\right). \end{aligned} \tag{C.24}$$

By fixing σ_y and σ_f in the term $\alpha_c = \frac{\sigma_y^2}{\sigma_f^2}\frac{|\mathcal{D}_c|}{|\mathcal{D}|}$, we obtain the familiar formulation of the PINN for the forward problem, i.e., (3.16) (without the dt-loss and with $\mathcal{L}_x$ instead of $\mathcal{L}_y$).

Bibliography

[1] 'The Paris Agreement - Publication | UNFCCC'. (29th Nov. 2018), [Online]. Available: https://unfccc.int/documents/184656 (visited on 27/06/2023).

[2] 'World Energy Transitions Outlook 2022'. (2022), [Online]. Available: https://www.irena.org/Digital-Report/World-Energy-Transitions-Outlook-2022#page-0 (visited on 27/06/2023).

[3] P. Kundur, N. J. Balu and M. G. Lauby, *Power System Stability and Control* (The EPRI Power System Engineering Series). New York: McGraw-Hill, 1994, 1176 pp.

[4] P. W. Sauer and M. A. Pai, *Power System Dynamics and Stability*. Upper Saddle River, N.J: Prentice Hall, 1998, 357 pp.

[5] F. Milano, *Power System Modelling and Scripting* (Power Systems). Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. DOI: 10.1007/978-3-642-13669-6.

[6] P. Kundur, J. Paserba, V. Ajjarapu *et al.*, 'Definition and classification of power system stability IEEE/CIGRE joint task force on stability terms and definitions', *IEEE Transactions on Power Systems*, vol. 19, no. 3, pp. 1387–1401, Aug. 2004. DOI: 10.1109/TPWRS.2004.825981.

[7] N. Hatziargyriou, J. Milanovic, C. Rahmann *et al.*, 'Definition and Classification of Power System Stability – Revisited & Extended', *IEEE Transactions on Power Systems*, vol. 36, no. 4, pp. 3271–3281, Jul. 2021. DOI: 10.1109/TPWRS.2020.3041774.

[8] P. Panciatici, G. Bareux and L. Wehenkel, 'Operating in the Fog: Security Management Under Uncertainty', *IEEE Power and Energy Magazine*, vol. 10, no. 5, pp. 40–49, Sep. 2012. DOI: 10.1109/MPE.2012.2205318.

[9] Y. Yasuda, L. Bird, E. M. Carlini *et al.*, 'C-E (curtailment – Energy share) map: An objective and quantitative measure to evaluate wind and solar curtailment', *Renewable and Sustainable Energy Reviews*, vol. 160, p. 112 212, May 2022. DOI: 10.1016/j.rser.2022.112212.

[10] E. A. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*. New York, McGraw-Hill, 1955, 458 pp.

[11] E. Süli and D. F. Mayers, *An Introduction to Numerical Analysis*. Cambridge ; New York: Cambridge University Press, 2003, 433 pp.

[12] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*, 2nd ed. Cambridge University Press, Nov. 2008. DOI: 10.1017/CBO9780511995569.

[13] B. Stott, 'Power system dynamic response calculations', *Proceedings of the IEEE*, vol. 67, no. 2, pp. 219–241, Feb. 1979. DOI: 10.1109/PROC.1979.11233.

[14] F. Milano, 'An Open Source Power System Analysis Toolbox', *IEEE Transactions on Power Systems*, vol. 20, no. 3, pp. 1199–1206, Aug. 2005. DOI: 10.1109/TPWRS.2005.851911.

[15] P. Aristidou, D. Fabozzi and T. Van Cutsem, 'Dynamic Simulation of Large-Scale Power Systems Using a Parallel Schur-Complement-Based Decomposition Method', *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2561–2570, Oct. 2014. DOI: 10.1109/TPDS.2013.252.

[16] H. Cui, F. Li and K. Tomsovic, 'Hybrid Symbolic-Numeric Framework for Power System Modeling and Analysis', *IEEE Transactions on Power Systems*, vol. 36, no. 2, pp. 1373–1384, Mar. 2021. DOI: 10.1109/TPWRS.2020.3017019.

[17] *PSCAD*, Manitoba Hydro International Ltd. [Online]. Available: https://www.pscad.com (visited on 30/06/2023).

[18] *PowerFactory*, DIgSILENT. [Online]. Available: https://www.digsilent.de/en/ (visited on 30/06/2023).

[19] P. Aristidou, 'Time-domain simulation of large electric power systems using domain-decomposition and parallel processing methods', Université de Liège, Liège, Belgium, 2015.

[20] G. Gurrala, A. Dimitrovski, S. Pannala, S. Simunovic and M. Starke, 'Parareal in Time for Fast Power System Dynamic Simulations', *IEEE Transactions on Power Systems*, vol. 31, no. 3, pp. 1820–1830, May 2016. DOI: 10.1109/TPWRS.2015.2434833.

[21] Y. Liu and K. Sun, 'Solving Power System Differential Algebraic Equations Using Differential Transformation', *IEEE Transactions on Power Systems*, vol. 35, no. 3, pp. 2289–2299, May 2020. DOI: 10.1109/TPWRS.2019.2945512.

[22] B. Wang, N. Duan and K. Sun, 'A Time–Power Series-Based Semi-Analytical Approach for Power System Simulation', *IEEE Transactions on Power Systems*, vol. 34, no. 2, pp. 841–851, Mar. 2019. DOI: 10.1109/TPWRS.2018.2871425.

[23] B. Park, K. Sun, A. Dimitrovski, Y. Liu and S. Simunovic, 'Examination of Semi-Analytical Solution Methods in the Coarse Operator of Parareal Algorithm for Power System Simulation', *IEEE Transactions on Power Systems*, vol. 36, no. 6, pp. 5068–5080, Nov. 2021. DOI: 10.1109/TPWRS.2021.3069136.

[24] I. Konstantelos, G. Jamgotchian, S. H. Tindemans, P. Duchesne, S. Cole, C. Merckx, G. Strbac and P. Panciatici, 'Implementation of a Massively Parallel Dynamic Security Assessment Platform for Large-Scale Grids', *IEEE Transactions on Smart Grid*, vol. 8, no. 3, pp. 1417–1426, May 2017. DOI: 10.1109/TSG.2016.2606888.

[25] L. Duchesne, E. Karangelos and L. Wehenkel, 'Recent Developments in Machine Learning for Energy Systems Reliability Management', *Proceedings of the IEEE*, vol. 108, no. 9, pp. 1656–1676, Sep. 2020. DOI: 10.1109/JPROC.2020.2988715.

[26] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations', *Journal of Computational Physics*, vol. 378, no. C, Nov. 2018. DOI: 10.1016/j.jcp.2018.10.045.

[27] G. S. Misyris, A. Venzke and S. Chatzivasileiadis, 'Physics-Informed Neural Networks for Power Systems', in *2020 IEEE Power & Energy Society General Meeting (PESGM)*, Montreal, QC, Canada: IEEE, 2020, pp. 1–5. DOI: 10.1109/PESGM41954.2020.9282004.

[28] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi and F. Piccialli, 'Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next', *Journal of Scientific Computing*, vol. 92, no. 3, p. 88, Jul. 2022. DOI: 10.1007/s10915-022-01939-z.

[29] B. Huang and J. Wang, 'Applications of Physics-Informed Neural Networks in Power Systems - A Review', *IEEE Transactions on Power Systems*, vol. 38, no. 1, pp. 572–588, Jan. 2023. DOI: 10.1109/TPWRS.2022.3162473.

[30] J. Willard, X. Jia, S. Xu, M. Steinbach and V. Kumar, 'Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems', *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–37, Apr. 2023. DOI: 10.1145/3514228.

[31] S. H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*, Second edition. Boulder, CO: Westview Press, a member of the Perseus Books Group, 2015, 513 pp.

[32] S. Greydanus, M. Dzamba and J. Yosinski, 'Hamiltonian neural networks', in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

[33] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel and S. Ho, 'Lagrangian Neural Networks', presented at the ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020. [Online]. Available: https://openreview.net/forum?id=iE8tFa4Nq.

[34] V. Grimm, A. Heinlein, A. Klawonn, M. Lanser and J. Weber, 'Estimating the time-dependent contact rate of SIR and SEIR models in mathematical epidemiology using physics-informed neural networks', *ETNA - Electronic Transactions on Numerical Analysis*, vol. 56, pp. 1–27, 2021. DOI: 10.1553/etna_vol56s1.

[35] J. Long, A. Q. M. Khaliq and K. M. Furati, 'Identification and prediction of time-varying parameters of COVID-19 model: A data-driven deep learning approach', *International Journal of Computer Mathematics*, vol. 98, no. 8, pp. 1617–1632, Aug. 2021. DOI: 10.1080/00207160.2021.1929942.

[36] S. Berkhahn and M. Ehrhardt, 'A physics-informed neural network to model COVID-19 infection and hospitalization scenarios', *Advances in Continuous and Discrete Models*, vol. 2022, no. 1, p. 61, Oct. 2022. DOI: 10.1186/s13662-022-03733-5.

[37] A. Rodríguez, J. Cui, N. Ramakrishnan, B. Adhikari and B. A. Prakash, 'EINNs: Epidemiologically-Informed Neural Networks', *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 12, pp. 14 453–14 460, Jun. 2023. DOI: 10.1609/aaai.v37i12.26690.

[38] R. Majumdar, S. Karande and L. Vig, 'DeepEpiSolver: Unravelling Inverse Problems in Covid, HIV, Ebola and Disease Transmission', presented at the 2023 ICLR First Workshop on Machine Learning & Global Health, 2023. [Online]. Available: https://openreview.net/forum?id=U5AiUCEDhy.

[39] K. Linka, A. Schäfer, X. Meng, Z. Zou, G. E. Karniadakis and E. Kuhl, 'Bayesian Physics Informed Neural Networks for real-world nonlinear dynamical systems', *Computer Methods in Applied Mechanics and Engineering*, vol. 402, p. 115 346, Dec. 2022. DOI: 10.1016/j.cma.2022.115346.

[40] M. A. Roehrl, T. A. Runkler, V. Brandtstetter, M. Tokic and S. Obermayer, 'Modeling System Dynamics with Physics-Informed Neural Networks Based on Lagrangian Mechanics', *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 9195–9200, 2020. DOI: 10.1016/j.ifacol.2020.12.2182.

[41] E. A. Antonelo, E. Camponogara, L. O. Seman, E. R. de Souza, J. P. Jordanou and J. F. Hubner. 'Physics-Informed Neural Nets for Control of Dynamical Systems'. arXiv: 2104.02556 [cs]. (31st May 2022), preprint.

[42] W. Ji, W. Qiu, Z. Shi, S. Pan and S. Deng, 'Stiff-PINN: Physics-Informed Neural Network for Stiff Chemical Kinetics', *The Journal of Physical Chemistry A*, vol. 125, no. 36, pp. 8098–8106, Sep. 2021. DOI: 10.1021/acs.jpca.1c05102.

[43] M. De Florio, E. Schiassi and R. Furfaro, 'Physics-informed neural networks and functional interpolation for stiff chemical kinetics', *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 32, no. 6, p. 063 107, Jun. 2022. DOI: 10.1063/5.0086649.

[44] Y. Weng and D. Zhou, 'Multiscale Physics-Informed Neural Networks for Stiff Chemical Kinetics', *The Journal of Physical Chemistry A*, vol. 126, no. 45, pp. 8534–8543, Nov. 2022. DOI: 10.1021/acs.jpca.2c06513.

[45] G. S. Gusmão, A. P. Retnanto, S. C. D. Cunha and A. J. Medford, 'Kinetics-informed neural networks', *Catalysis Today*, vol. 417, p. 113 701, May 2023. DOI: 10.1016/j.cattod.2022.04.002.

[46] L. Yang, X. Meng and G. E. Karniadakis, 'B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data', *Journal of Computational Physics*, vol. 425, p. 109 913, Jan. 2021. DOI: 10.1016/j.jcp.2020.109913.

[47] T. M. Mitchell, *Machine Learning* (McGraw-Hill Series in Computer Science). New York: McGraw-Hill, 1997, 414 pp.

[48] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning* (Springer Series in Statistics). New York, NY: Springer New York, 2009. DOI: 10.1007/978-0-387-84858-7.

[49] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning* (Adaptive Computation and Machine Learning). Cambridge, Massachusetts: The MIT Press, 2016, 775 pp.

[50] J. Z. Kolter and T. Chen, '10-414/714: Deep Learning Systems', (CMU), 2022. [Online]. Available: https://dlsyscourse.org/lectures/ (visited on 20/05/2023).

[51] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge: Cambridge University Press, 2019.

[52] J. Guckenheimer, 'Numerical Analysis of Dynamical Systems', in *Handbook of Dynamical Systems*, vol. 2, Elsevier, 2002, pp. 345–390. DOI: 10.1016/S1874-575X(02)80029-7.

[53] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields* (Applied Mathematical Sciences). New York, NY: Springer New York, 1983, vol. 42. DOI: 10.1007/978-1-4612-1140-2.

[54] A. M. Stuart and A. R. Humphries, *Dynamical Systems and Numerical Analysis* (Cambridge Monographs on Applied and Computational Mathematics 2), 1. paperback ed. Cambridge, United Kingdom: Cambridge University Press, 1998, 685 pp.

[55] A. M. Stuart, 'Numerical analysis of dynamical systems', *Acta Numerica*, vol. 3, pp. 467–572, Jan. 1994. DOI: 10.1017/S0962492900002488.

[56] K. Hornik, M. Stinchcombe and H. White, 'Multilayer feedforward networks are universal approximators', *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989. DOI: 10.1016/0893-6080(89)90020-8.

[57] G. Cybenko, 'Approximation by superpositions of a sigmoidal function', *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989. DOI: 10.1007/BF02551274.

[58] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, 'Backpropagation Applied to Handwritten Zip Code Recognition', *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989. DOI: 10.1162/neco.1989.1.4.541.

[59] D. Rumelhart, G. Hinton and R. Williams, 'Learning Internal Representations by Error Propagation', in *Readings in Cognitive Science*, Elsevier, 1988, pp. 399–421. DOI: 10.1016/B978-1-4832-1446-7.50035-2.

[60] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner and G. Monfardini, 'The Graph Neural Network Model', *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, Jan. 2009. DOI: 10.1109/TNN.2008.2005605.

[61] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming* (International Series in Operations Research and Management Science), 3rd ed. New York, NY: Springer, 2008, 546 pp.

[62] D. P. Kingma and J. Ba, 'Adam: A Method for Stochastic Optimization', presented at the International Conference on Learning Representations, San Diego, 2015. arXiv: 1412.6980 [cs].

[63] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak and I. Sutskever, 'Deep Double Descent: Where Bigger Models and More Data Hurt', presented at the International Conference on Learning Representations, 2020. [Online]. Available: https://openreview.net/forum?id=B1g5sA4twr.

[64] A. G. Wilson and P. Izmailov, 'Bayesian Deep Learning and a Probabilistic Perspective of Generalization', in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 4697–4708. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/322f62469c5e3c7dc3e58f5a4d1ea399-Paper.pdf.

[65] S. Gunasekar, J. D. Lee, D. Soudry and N. Srebro, 'Implicit Bias of Gradient Descent on Linear Convolutional Networks', in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/0e98aeeb54acf612b9eb4e48a269814c-Paper.pdf.

[66] N. Baker, F. Alexander, T. Bremer *et al.*, 'Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence', 1478744, 10th Feb. 2019. DOI: 10.2172/1478744.

[67] I. Lagaris, A. Likas and D. Fotiadis, 'Artificial neural networks for solving ordinary and partial differential equations', *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, Sep. 1998. DOI: 10.1109/72.712178.

[68] A. Paszke, S. Gross, F. Massa *et al.*, 'PyTorch: An imperative style, high-performance deep learning library', in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[69] M. Abadi, A. Agarwal, P. Barham *et al.*, 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems', 16th Mar. 2016. arXiv: 1603.04467 [cs].

[70] J. Bradbury, R. Frostig, P. Hawkins *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.3.13, Google, 2018. [Online]. Available: https://github.com/google/jax.

[71] M. Innes, 'Flux: Elegant machine learning with Julia', *Journal of Open Source Software*, vol. 3, no. 25, p. 602, May 2018. DOI: 10.21105/joss.00602. [Online]. Available: http://joss.theoj.org/papers/10.21105/joss.00602.

[72] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal and V. Shah. 'Fashionable Modelling with Flux'. arXiv: 1811.01457 [cs]. (Nov. 2018), preprint.

[73] A. G. Baydin, B. A. Pearlmutter, A. A. Radul and J. M. Siskind, 'Automatic Differentiation in Machine Learning: A Survey', *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018.

[74] G. I. Barenblatt, *Scaling, Self-similarity, and Intermediate Asymptotics: Dimensional Analysis and Intermediate Asymptotics*, 1st ed. Cambridge University Press, 12th Dec. 1996. DOI: 10.1017/CBO9781107050242.

[75] M. A. Pai, *Energy Function Analysis for Power System Stability*. Boston, MA: Springer US, 1989.

[76] X. Glorot and Y. Bengio, 'Understanding the difficulty of training deep feedforward neural networks', in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html.

[77] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, 'Understanding deep learning requires rethinking generalization', presented at the International Conference on Learning Representations, 2017. [Online]. Available: https://openreview.net/forum?id=Sy8gdB9xx.

[78] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy and P. T. P. Tang, 'On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima', presented at the International Conference on Learning Representations, 2017. [Online]. Available: https://openreview.net/forum?id=H1oyRlYgg.

[79] S. Wang, Y. Teng and P. Perdikaris, 'Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks', *SIAM Journal on Scientific Computing*, vol. 43, no. 5, A3055–A3081, Jan. 2021. DOI: 10.1137/20M1318043.

[80] S. Wang, S. Sankaran and P. Perdikaris. 'Respecting causality is all you need for training physics-informed neural networks'. arXiv: 2203.07404 [cs.LG]. (Mar. 2022), preprint.

[81] M. M. Bronstein, J. Bruna, T. Cohen and P. Veličković. 'Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges'. arXiv: 2104.13478 [cs.LG]. (Apr. 2021), preprint.

[82] N. Doumèche, G. Biau and C. Boyer. 'Convergence and error analysis of PINNs'. arXiv: 2305.01240 [math.ST]. (May 2023), preprint.

[83] Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su and J. Zhu. 'Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications'. arXiv: 2211.08064 [cs, math]. (Mar. 2023), preprint.

[84] S. A. Faroughi, N. Pawar, C. Fernandes, M. Raissi, S. Das, N. K. Kalantari and S. K. Mahjour. 'Physics-Guided, Physics-Informed, and Physics-Encoded Neural Networks in Scientific Computing'. arXiv: 2211.07377 [cs]. (Feb. 2023), preprint.

[85] C. Legaard, T. Schranz, G. Schweiger, J. Drgoňa, B. Falay, C. Gomes, A. Iosifidis, M. Abkar and P. Larsen, 'Constructing Neural Network Based Models for Simulating Dynamical Systems', *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–34, 2023. DOI: 10.1145/3567591.

[86] S. Hochreiter and J. Schmidhuber, 'Long Short-Term Memory', *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[87] B. Chang, M. Chen, E. Haber and E. H. Chi, 'AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks', presented at the International Conference on Learning Representations, 2018. [Online]. Available: https://openreview.net/forum?id=ryxepo0cFX.

[88] C. Tallec and Y. Ollivier, 'Can recurrent neural networks warp time?', presented at the International Conference on Learning Representations, 2018. [Online]. Available: https://openreview.net/forum?id=72xO7jUzvCs.

[89] L. Lu, P. Jin, G. Pang, Z. Zhang and G. E. Karniadakis, 'Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators', *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, Mar. 2021. DOI: 10.1038/s42256-021-00302-5.

[90] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart and A. Anandkumar, 'Fourier Neural Operator for Parametric Partial Differential Equations', presented at the International Conference on Learning Representations, 2021. [Online]. Available: https://openreview.net/forum?id=c8P9NQVtmnO.

[91] S. Wang and P. Perdikaris, 'Long-time integration of parametric evolution equations with physics-informed DeepONets', *Journal of Computational Physics*, vol. 475, p. 111 855, Feb. 2023. DOI: 10.1016/j.jcp.2022.111855.

[92] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli and A. Anandkumar. 'Physics-Informed Neural Operator for Learning Partial Differential Equations'. arXiv: 2111.03794 [cs.LG]. (Apr. 2023), preprint.

[93] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart and A. Anandkumar, 'Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs', *Journal of Machine Learning Research*, vol. 24, no. 89, pp. 1–97, 2023. [Online]. Available: http://jmlr.org/papers/v24/21-1524.html.

[94] T. Chen and H. Chen, 'Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems', *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, Jul. 1995. DOI: 10.1109/72.392253.

[95] N. Kovachki, S. Lanthaler and S. Mishra, 'On Universal Approximation and Error Bounds for Fourier Neural Operators', *Journal of Machine Learning Research*, vol. 22, no. 290, pp. 1–76, 2021. [Online]. Available: http://jmlr.org/papers/v22/21-0806.html.

[96] T. De Ryck and S. Mishra, 'Generic bounds on the approximation error for physics-informed (and) operator learning', in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 10 945–10 958. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/46f0114c06524debc60ef2a72769f7a9-Paper-Conference.pdf.

[97] Y. Shin, J. Darbon and G. E. Karniadakis, 'On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs', *Communications in Computational Physics*, vol. 28, no. 5, pp. 2042–2074, Jun. 2020. DOI: 10.4208/cicp.OA-2020-0193.

[98] B. O. Koopman, 'Hamiltonian Systems and Transformation in Hilbert Space', *Proceedings of the National Academy of Sciences*, vol. 17, no. 5, pp. 315–318, May 1931. DOI: 10.1073/pnas.17.5.315.

[99] S. L. Brunton, M. Budišić, E. Kaiser and J. N. Kutz, 'Modern Koopman Theory for Dynamical Systems', *SIAM Review*, vol. 64, no. 2, pp. 229–340, May 2022. DOI: 10.1137/21M1401243.

[100] R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. K. Duvenaud, 'Neural Ordinary Differential Equations', in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[101] A. S. Krishnapriyan, A. F. Queiruga, N. B. Erichson and M. W. Mahoney. 'Learning continuous models for continuous physics'. arXiv: 2202.08494 [cs.LG]. (Feb. 2022), preprint.

[102] L. Ljung, *System Identification: Theory for the User* (Prentice Hall Information and System Sciences Series), 2nd ed. Upper Saddle River, NJ: Prentice Hall PTR, 1999, 609 pp.

[103] A. Gelman, *Bayesian Data Analysis* (Chapman & Hall/CRC Texts in Statistical Science), Third edition. Boca Raton: CRC Press, 2014, 661 pp.

[104] H. Raiffa and R. Schlaifer, *Applied Statistical Decision Theory* (Wiley Classics Library), Wiley classics library ed. New York: Wiley, 2000, 356 pp.

[105] S. Brooks, A. Gelman, G. Jones and X.-L. Meng, Eds., *Handbook of Markov Chain Monte Carlo*. New York: Chapman and Hall/CRC, 24th May 2011, 619 pp. DOI: 10.1201/b10905.

[106] M. D. Hoffman and A. Gelman, 'The no-U-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo', *Journal of Machine Learning Research*, vol. 15, no. 47, pp. 1593–1623, 2014. [Online]. Available: http://jmlr.org/papers/v15/hoffman14a.html.

[107] D. Phan, N. Pradhan and M. Jankowiak. 'Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro'. arXiv: 1912.11554 [stat.ML]. (Dec. 2019), preprint.

[108] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall and N. D. Goodman, 'Pyro: Deep Universal Probabilistic Programming', *Journal of Machine Learning Research*, vol. 20, no. 28, pp. 1–6, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-403.html.

[109] M. D. Hoffman, D. M. Blei, C. Wang and J. Paisley, 'Stochastic Variational Inference', *Journal of Machine Learning Research*, vol. 14, no. 40, pp. 1303–1347, 2013. [Online]. Available: http://jmlr.org/papers/v14/hoffman13a.html.

[110] Q. Liu and D. Wang, 'Stein variational gradient descent: A general purpose bayesian inference algorithm', in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/b3ba8f1bee1238a2f37603d90b58898d-Paper.pdf.

[111] D. J. C. Mackay, 'Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks', *Network: Computation in Neural Systems*, vol. 6, no. 3, pp. 469–505, Jan. 1995. DOI: 10.1088/0954-898X_6_3_011.

[112] D. J. C. MacKay, 'A Practical Bayesian Framework for Backpropagation Networks', *Neural Computation*, vol. 4, no. 3, pp. 448–472, May 1992. DOI: 10.1162/neco.1992.4.3.448.

[113] C. Blundell, J. Cornebise, K. Kavukcuoglu and D. Wierstra, 'Weight Uncertainty in Neural Network', in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, 1st Jun. 2015, pp. 1613–1622. [Online]. Available: https://proceedings.mlr.press/v37/blundell15.html.

[114] T. Papamarkou, J. Hinkle, M. T. Young and D. Womble, 'Challenges in Markov Chain Monte Carlo for Bayesian Neural Networks', *Statistical Science*, vol. 37, no. 3, 1st Aug. 2022. DOI: 10.1214/21-STS840.

[115] S. Perez, S. Maddu, I. F. Sbalzarini and P. Poncet. 'Adaptive weighting of Bayesian physics informed neural networks for multitask and multiscale forward and inverse problems'. arXiv: 2302.12697 [physics.comp-ph]. (Feb. 2023), preprint.

[116] S. L. Brunton, J. L. Proctor and J. N. Kutz, 'Discovering governing equations from data by sparse identification of nonlinear dynamical systems', *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 12th Apr. 2016. DOI: 10.1073/pnas.1517384113.

[117] U. Fasel, J. N. Kutz, B. W. Brunton and S. L. Brunton, 'Ensemble-SINDy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control', *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 478, no. 2260, p. 20 210 904, Apr. 2022. DOI: 10.1098/rspa.2021.0904.

[118] P. Kidger, 'On Neural Differential Equations', University of Oxford, Oxford, United Kingdom, 2022. arXiv: 2202.02435 [cs.LG].

[119] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan and A. Edelman. 'Universal Differential Equations for Scientific Machine Learning'. arXiv: 2001.04385 [cs.LG]. (Nov. 2021), preprint.

[120] H. Dommel, 'Digital Computer Solution of Electromagnetic Transients in Single-and Multiphase Networks', *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-88, no. 4, pp. 388–399, Apr. 1969. DOI: 10.1109/TPAS.1969.292459.

[121] C. Gear, 'Simultaneous Numerical Solution of Differential-Algebraic Equations', *IEEE Transactions on Circuit Theory*, vol. 18, no. 1, pp. 89–95, 1971. DOI: 10.1109/TCT.1971.1083221.

[122] L. Petzold, 'Differential/Algebraic Equations are not ODE' S', *SIAM Journal on Scientific and Statistical Computing*, vol. 3, no. 3, pp. 367–384, Sep. 1982. DOI: 10.1137/0903023.

[123] K. E. Brenan, S. L. Campbell and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, Jan. 1995. DOI: 10.1137/1.9781611971224.

[124] G. Adomian, 'A review of the decomposition method in applied mathematics', *Journal of Mathematical Analysis and Applications*, vol. 135, no. 2, pp. 501–544, Nov. 1988. DOI: 10.1016/0022-247X(88)90170-9.

[125] G. Gurrala, D. L. Dinesha, A. Dimitrovski, P. Sreekanth, S. Simunovic and M. Starke, 'Large Multi-Machine Power System Simulations Using Multi-Stage Adomian Decomposition', *IEEE Transactions on Power Systems*, vol. 32, no. 5, pp. 3594–3606, Sep. 2017. DOI: 10.1109/TPWRS.2017.2655300.

[126] C. Moya, G. Lin, T. Zhao and M. Yue. 'On Approximating the Dynamic Response of Synchronous Generators via Operator Learning: A Step Towards Building Deep Operator-based Power Grid Simulators'. arXiv: 2301.12538 [cs.LG]. (Jan. 2023), preprint.

[127] C. Moya, S. Zhang, G. Lin and M. Yue, 'DeepONet-grid-UQ: A trustworthy deep operator framework for predicting the power grid's post-fault trajectories', *Neurocomputing*, vol. 535, pp. 166–182, May 2023. DOI: 10.1016/j.neucom.2023.03.015.

[128] W. Cui, W. Yang and B. Zhang. 'Predicting Power System Dynamics and Transients: A Frequency Domain Approach'. arXiv: 2111.01103 [eess.SY]. (Nov. 2021), preprint.

[129] C. Moya and G. Lin, 'DAE-PINN: A physics-informed neural network model for simulating differential algebraic equations with application to power networks', *Neural Computing and Applications*, vol. 35, no. 5, pp. 3789–3804, Feb. 2023. DOI: 10.1007/s00521-022-07886-y.

[130] L. Pagnier, J. Fritzsch, P. Jacquod and M. Chertkov, 'Toward Model Reduction for Power System Transients With Physics-Informed PDE', *IEEE Access*, vol. 10, pp. 65 118–65 125, 2022. DOI: 10.1109/ACCESS.2022.3183336.

[131] C. Andersson, C. Führer and J. Åkesson, 'Assimulo: A unified framework for ODE solvers', *Mathematics and Computers in Simulation*, vol. 116, pp. 26–43, Oct. 2015. DOI: 10.1016/j.matcom.2015.04.007.

[132] P. M. Anderson and A. A. Fouad, *Power System Control and Stability* (IEEE Press Power Engineering Series), 2nd ed. Piscataway, N.J: IEEE Press ; Wiley-Interscience, 2003, 658 pp.

[133] R. D. Zimmerman, C. E. Murillo-Sanchez and R. J. Thomas, 'MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education', *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, Feb. 2011. DOI: 10.1109/TPWRS.2010.2051168.

[134] B. Donon, B. Donnot, I. Guyon and A. Marot, 'Graph Neural Solver for Power Systems', in *2019 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851855.

[135] B. Donon, R. Clément, B. Donnot, A. Marot, I. Guyon and M. Schoenauer, 'Neural networks for power flow: Graph neural solver', *Electric Power Systems Research*, vol. 189, p. 106 547, 1st Dec. 2020. DOI: 10.1016/j.epsr.2020.106547.

[136] K. Baker, 'Emulating AC OPF Solvers With Neural Networks', *IEEE Transactions on Power Systems*, vol. 37, no. 6, pp. 4950–4953, Nov. 2022. DOI: 10.1109/TPWRS.2022.3195097.

[137] B. Dembart, A. M. Erisman, E. G. Cate, M. A. Epton and H. Dommel, 'Power system dynamic analysis: Phase I. Final report', EPRI-EL-484, 7296152, 1977. DOI: 10.2172/7296152.

[138] R. T. Q. Chen, *Torchdiffeq*, 2018. [Online]. Available: https://github.com/rtqichen/torchdiffeq.

[139] C. Rackauckas and Q. Nie, 'DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia', *Journal of Open Research Software*, vol. 5, no. 1, p. 15, 25th May 2017. DOI: 10.5334/jors.151.

[140] J. Scott and M. Tůma, *Algorithms for Sparse Linear Systems* (Nečas Center Series). Cham: Springer International Publishing, 2023. DOI: 10.1007/978-3-031-25820-6.

[141] T. Weckesser, H. Johannsson, S. Sommer and J. Ostergaard, 'Investigation of the adaptability of transient stability assessment methods to real-time operation', in *2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*, Berlin, Germany: IEEE, Oct. 2012, pp. 1–9. DOI: 10.1109/ISGTEurope.2012.6465835.

[142] D. Silver, A. Huang, C. J. Maddison *et al.*, 'Mastering the game of Go with deep neural networks and tree search', *Nature*, vol. 529, no. 7587, pp. 484–489, 28th Jan. 2016. DOI: 10.1038/nature16961.

[143] B. Yu and K. Kumbier, 'Veridical data science', *Proceedings of the National Academy of Sciences*, vol. 117, no. 8, pp. 3920–3929, 25th Feb. 2020. DOI: 10.1073/pnas.1901326117. pmid: 32054788.

[144] A. Paleyes, R.-G. Urma and N. D. Lawrence, 'Challenges in Deploying Machine Learning: A Survey of Case Studies', *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–29, 31st Jul. 2023. DOI: 10.1145/3533378.

[145] F. Pedregosa, G. Varoquaux, A. Gramfort *et al.*, 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html.

[146] L. Biewald, *Experiment Tracking with Weights and Biases*, 2020. [Online]. Available: https://www.wandb.com/.

[147] *PyTorch Lightning*, Lightning AI et al. [Online]. Available: https://www.pytorchlightning.ai (visited on 27/06/2023).

[148] *Hugging Face – The AI community building the future.* [Online]. Available: https://huggingface.co/.

[149] 'Formal Verification of Deep Neural Networks: Theory and Practice - Tutorial'. (), [Online]. Available: https://neural-network-verification.com/ (visited on 27/06/2023).

[150] D. C. Center, *DTU Computing Center resources*, Technical University of Denmark, 2022. [Online]. Available: https://doi.org/10.48714/DTU.HPC.0001 (visited on 19/08/2022).

Department of Wind and Energy Systems
Division for Power and Energy Systems (PES)
Technical University of Denmark
Elektrovej, Building 325
DK-2800 Kgs. Lyngby
Denmark