Closing the Sim-to-Real Gap with Physics-enhanced Neural ODEs

Tobias Kamp[1][a], Johannes Ultsch[1][b] and Jonathan Brembeck[1][c]
[1]*German Aerospace Center, Institute of System Dynamics and Control (SR)*
tobias.kamp@dlr.de

Keywords: Dynamical Systems, Hybrid Modelling, Neural Ordinary Differential Equations, Scientific Machine Learning, Physics-enhanced Neural ODEs

Abstract: A central task in engineering is the modelling of dynamical systems. In addition to first-principle methods, data-driven approaches leverage recent developments in machine learning to infer models from observations. Hybrid models aim to inherit the advantages of both, white- and black-box modelling approaches by combining the two methods in various ways. In this sense, Neural Ordinary Differential Equations (NODEs) proved to be a promising approach that deploys state-of-the-art ODE solvers and offers great modelling flexibility. In this work, an exemplary NODE setup is used to train low-dimensional artificial neural networks with physically meaningful outputs to enhance a dynamical model. The approach maintains the physical integrity of the model and offers the possibility to enforce physical laws during the training. Further, this work outlines how a confidence interval for the learned functions can be inferred based on the deployed training data. The robustness of the approach against noisy data and model uncertainties is investigated and a way to optimize model parameters alongside the neural networks is shown. Finally, the training routine is optimized with mini-batching and sub-sampling, which reduces the training duration in the given example by over 80 %.

1 INTRODUCTION

The modelling of dynamical systems is an important and challenging engineering task which forms the foundation for subsequent controller design, optimization, visualization and many more. Models are usually optimized for specific applications, since *exact* (white-box) modelling of complex systems soon becomes infeasible. In order to optimize a model regarding its prediction quality or computational complexity, conventional data-driven approaches like parameter optimization, system identification and the usage of look-up tables are common practice.

Black-box Modelling

With the rise of machine learning (ML) algorithms and toolboxes, manifold data-driven modelling approaches for dynamical systems emerged. Especially recurrent neural networks (RNNs) proved to be well suited to learn time-dependent correlations (Haber and Ruthotto, 2017), (Chang et al., 2019). Neural Ordinary Differential Equations (NODEs), as introduced by Chen (Chen et al., 2018), pose another promising approach. NODEs only approximate the right-hand side of the differential equations with neural networks (NNs) and benefit from the usage of well-established ODE solvers that enable time-continuous simulation and the handling of stiff systems and time events. However, pure black-box approaches generally suffer from poor extrapolation, high data demand and instability. Knowledge of the fundamental dynamics of a system can be used to mitigate these disadvantages using a hybrid modelling approach.

Hybrid Modelling

Hybrid models combine first-principle methods with machine learning and aim to leverage the advantages of both. One way to use the existing knowledge is to enforce physically meaningful behaviour of the black-box model during the training. Raissi introduced Physics-informed neural networks (PINNs) (Raissi et al., 2019) to solve problems that involve partial differential equations. These networks leverage the power of automatic differentiation to incorporate knowledge about the derivatives into the loss function. Another way to use the physical equations is the explicit combination with black-box components.

[a] https://orcid.org/0009-0006-5584-2928
[b] https://orcid.org/0000-0001-6483-8468
[c] https://orcid.org/0000-0002-7671-5251

In this case, the white-box part serves e.g. to pre-process the inputs, initialize hidden states of recurrent architectures or to provide an a-priori estimation that is afterwards corrected by data-driven components (cf. residual-physics, (Daw et al., 2022), (Zeng et al., 2020)). The idea of residual-physics formed the basis for dedicated hybrid simulators which allow the incorporation and training of NNs in physics-based models (Ajay et al., 2018), (Heiden et al., 2020). For a more complete overview of current developments in hybrid modelling, we refer the interested reader to the surveys of Willard (Willard et al., 2020), Rai (Rai and Sahu, 2020) and Karniadakis (Karniadakis et al., 2021).

Universal Differential Equations

Universal differential equations (UDEs) (Rackauckas et al., 2020) expand the basic idea of Neural ODEs and allow arbitrary designs of the differential equations. This enables the enrichment of dynamical models with NNs to expand or replace equations or computational costly parts. The concept was used in several works and in different domains, for instance in vehicle dynamics (Bruder and Mikelsons, 2021), (Thummerer et al., 2022), chemistry (Owoyele and Pal, 2022), climate modelling (Ramadhan et al., 2022) or fluid dynamics (Thummerer et al., 2021).

Contribution

The generality and physical integrity of a hybrid model diminishes with increasing influence and complexity of its black-box components, while the demand for training data rises. In the reviewed applications of NODEs, the physical model is either combined with data-driven components to form a higher-order model (Thummerer et al., 2021), (Thummerer et al., 2022) or the NNs approximate the right-hand side of single differential equations (Bruder and Mikelsons, 2021), (Owoyele and Pal, 2022). While the obtained models demonstrate the potentials of hybrid approaches, y NODEs are prone to instability issues and/or local optima (Turan and Jaschke, 2022) and the analysis of the physical feasibility and extrapolation is not trivial. Inspired by the idea of "fine-grained data-driven models" (Heiden et al., 2020), we propose to go one step deeper and insert *physically meaningful neural components* into the given differential equations, forming a Physics-enhanced NODE (PeNODE). This has the advantage, that the learned functions can be analysed directly and physical properties can be enforced during the training. In addition, we show how a confidence interval can be derived from the training data, which serves to define the application boundaries of the hybrid model. Another benefit of the presented approach is the limited influence of the data-driven components, which reduces stability and convergence issues and minimises the data-demand.

Outline

This work is structured as follows: In Section 2, we present the concept of PeNODEs and how the loss-function can be used to enforce physical laws. In Section 3, the training routine and the benefits of mini-batching and sub-sampling are described. Our demonstrator, a quarter vehicle model, is introduced in Section 4. The conducted experiments and corresponding results are presented in Section 5 before concluding with Section 6.

2 METHOD

This section presents a method to formulate PeNODEs and how to enforce properties with regularizing terms. Further, the idea of simultaneous parameter fitting is introduced.

2.1 Terminology

The meaningful combination of physical equations and NNs is referred to as Physics-enhanced Neural ODE (PeNODE). The additional usage of regularizing terms that guide the optimization and enforce certain properties would be a "Physics-Informed Physics-enhanced Neural ODE" (cf. PINNs, (Raissi et al., 2019)). For the sake of simplicity, we however assume that a Physics-enhanced Neural ODE can also be Physics-Informed.

2.2 Prerequisites

This work postulates the existence of a dynamical model, which fairly represents the fundamental dynamics of the considered system. The model must be given in the general form

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}, \boldsymbol{u}, t) \quad (1)$$

$$\boldsymbol{y}(t) = g(\boldsymbol{x}, \boldsymbol{u}, t) \; , \quad (2)$$

with a state-vector $\boldsymbol{x}$, input-vector $\boldsymbol{u}$ and output-vector $\boldsymbol{y}$.

2.3 Deriving a PeNODE

In order to obtain a neural ODE, the right-hand side of the differential equation (1) is enhanced with one or more NNs with parameters Θ:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}, \boldsymbol{u}, t, NN(\boldsymbol{x}, \boldsymbol{u}, t, \Theta)) \ . \quad (3)$$

The analogy of systems in different domains (Hogan and Breedveld, 2005) offers the possibility to sketch a general approach to obtain a PeNODE, i.e. a meaningful combination of the physical equations and NNs:

In the context of systems analogy, the three basic components of a dynamical system are compliance, inductance and resistance. Further, a domain-specific pair of power conjugated variables, i.e. one *flow* variable and one *potential* variable can be defined. Component-specific (differential-) equations correlate flow and potential (e.g. $U = RI$ for an electrical resistance) and/or define the transition to another domain. The differential equations of the system can be derived by setting the potentials to be equal and the sum of flows equal to zero for each connection of the components and substitute the component-specific equations (Zimmer, 2016).

Based on the described formalism, we propose to introduce *neural components* into the systems equations that obey the same fundamental rules and learn the correlation of flow and potential. The number of relevant inputs for these neural components is expected to be low, since they capture but one aspect of the complex system they compose. The NNs thus have a low-dimensional input and an one-dimensional output. This “fine-grained” approach is advantageous, because the obtained neural components are easy to analyse regarding their physical feasibility and extrapolation properties. Furthermore, physically meaningful behaviour can be encouraged during the training by punishing unphysical outputs. Finally, the limited influence of the black-boxes avoids stability issues and minimises the data demand.

The presented approach is well suited to enhance models which already capture the fundamental dynamics of the real system. In this case, the neural components can account for secondary effects like friction, elasticities or non-linear characteristics that cause the *Sim-to-Real Gap*, i.e. the residuum between model predictions and real observations.

2.4 Physics-informed Training

During the training, the parameters of the neural components are adapted to minimise the error (or loss) of the model-predictions with respect to given training data. In order to obtain physically reasonable neural components, regularizing terms that punish unphysical behaviour can be added to the loss function. For example, zero-crossing of learned functions is desirable in many cases: Consider a neural resistance (damper) with trainable parameters Θ which introduces a damper force $F_{R,NN}(v)$ that depends on a velocity v. For $v = 0$, this force must be equal to zero. Thus, a possible regularizing term is given by

$$r_{zcross}(\Theta) = F_{R,NN}(0, \Theta)^2 \ . \quad (4)$$

Since a regularizing term in the loss function cannot enforce *exact* zero-crossing, the remaining offset can be eliminated after the training by subtraction:

$$\tilde{F}_{R,NN}(v) = F_{R,NN}(v) - F_{R,NN}(0) \ . \quad (5)$$

If secondary dependencies, i.e. more inputs are added to the neural component, the concepts of regularizing and offset-correction are still applicable. In that case, the zero-crossing values of arbitrary sample points of the input space of the secondary inputs can be used for the regularization. In the given example, if a dependency of the temperature T is added, the regularizer can be adapted to

$$r_{zcross}(\Theta) = \frac{1}{N} \sum_{i=0}^{N} F_{R,NN}(0, T_i, \Theta)^2 \ , \quad (6)$$

with N samples from the temperature input-space.

The given example shows how a regularizer for a single neural component can be formulated. In addition to such specific terms, general knowledge about system properties like stability and oscillation capability can be incorporated using eigen-informed regularizers (Thummerer and Mikelsons, 2023).

2.5 Optimizing Model Parameters

The presented approach relies on low-dimensional NNs whose influence is limited by design. In order to add further degrees of freedom to the optimization, certain model parameters, e.g. the masses or inertia, can be optimized alongside the parameters of the NNs. To optimize model parameters $\{p_1, p_2, ...\}$, we propose to define a vector of parameter-modifiers $P_\Delta = [\delta p_1, \ \delta p_2, ...]^T$ that is concatenated with the vector of the NN-parameters and subjected to the gradient-based optimization. To ensure stability of the model and limit the impact of the modifiers, a smooth saturation with a hyperbolic tangent function can be used. The modified model parameters $\tilde{p}_i$ are then given by

$$\tilde{p}_i = (1 + \overline{\delta p_i}) p_i \quad (7)$$

$$\overline{\delta p_i} = \gamma \tanh(\delta p_i) \ . \quad (8)$$

The hyper-parameter $\gamma \in (0, 1)$ ensures that the parameters can only be changed by $\max. \pm \gamma$, e.g. $\pm 20\,\%$.

3 TRAINING DETAILS

In this section, the training process for a NODE is briefly sketched, revealing how mini-batching and sub-sampling help to reduce the computational complexity and how to use model outputs as physical features.

3.1 Basic Training

The basic training routine of a NODE is to repeatedly simulate the model for a given time horizon, calculate the loss with respect to all or a subset of the states and adapt the weights of the NNs in a step of a gradient-based algorithm, e.g. gradient descent. The loss function can be any common metric like the (root) mean squared error over all time-points and states. The gradient-based optimization requires the usage of a differentiable ODE-solver, as provided in Julia (Bezanson et al., 2017) or PyTorch[1]. This consequently assumes that the model is (re)implemented in the training environment. With `FMIFlux.jl`[2], the model exchange standard FMI[3] can be used to facilitate this process and integrate FMUs into Julia applications.

3.2 Mini-Batching and Sub-sampling

The computational complexity of the gradient calculation increases significantly with the length and sample rate of the deployed training trajectories. Mini-batching can be used to eliminate this effect and make the training scalable to arbitrary datasets by using a number of short sequences from the original dataset for each update. A further reduction of complexity can be achieved by sub-sampling, i.e. calculating the loss for a subset of the data points. To introduce randomness and avoid aliasing, the points can be sampled randomly instead of using equidistant time-intervals.

3.3 Model Outputs as Physical Features

In many cases, not all states of a system are measurable with reasonable effort. While methods for state estimation can be used to fill in missing measurements, the NODE can also be trained on a subset of states or other quantities. In this case, the model observation function (cf. Equation (2)) is used to transform the state trajectory into an output trajectory which reflects the available measurements. Using the model outputs for the training is a form of feature extraction and can serve to condense the possibly high-dimensional state-space to a set of (relevant) quantities. Note, that if measurement for states are missing, the usage of mini-batching is impeded, since the initial state for each sequence must be given to solve the differential equations correctly.

[1]https://github.com/rtqichen/torchdiffeq

[2]https://github.com/ThummeTo/FMIFlux.jl

[3]https://fmi-standard.org/

4 DEMONSTRATOR

In this section we introduce our demonstrator, a quarter vehicle model (QVM). A non-linear version of the QVM serves to generate training data and the linear QVM is transformed into a PeNODE that is used in several training-setups to demonstrate our method.

4.1 The Linear QVM

The QVM is commonly used to represent the vertical dynamics of road vehicles. It consists of two masses, one for the wheel (m_w) and one for one quarter of the body (m_b) (cf. Figure 1). The two masses are connected by a linear spring-damper pair with coefficients c_s and d_s, representing the vehicle suspension (index s). Another linear spring-damper pair with coefficients c_t and d_t represents the tire that connects the vehicle to the ground (index t). The state vector $\boldsymbol{x}$ consists of the road height z_r, the position and velocity of the wheel z_w and v_w and the position and velocity of the body z_b and v_b. The input $u(t)$ is the differential road height $\dot{z}_r$ and the model output $\boldsymbol{y}$ contains the accelerations of the wheel and the body a_w and a_b.

$$\boldsymbol{x} = [x_1, ..., x_5]^T = [z_b,\ v_b,\ z_w,\ v_w,\ z_r]^T \tag{9}$$

$$\boldsymbol{y} = [\dot{x}_2,\ \dot{x}_4]^T = [a_b,\ a_w]^T \tag{10}$$

$$u = \dot{z}_r \tag{11}$$

The linear differential equations that describe the system (12 - 16) can be derived using the Newtonian laws of motion.

$$\dot{x}_1 = v_b \tag{12}$$

$$\dot{x}_2 = m_b^{-1}(c_s \Delta z_s + d_s \Delta v_s) \tag{13}$$

$$\dot{x}_3 = v_w \tag{14}$$

$$\begin{aligned}\dot{x}_4 = m_w^{-1}(&c_t \Delta z_t + d_t \Delta v_t \\ &- c_s \Delta z_s - d_s \Delta v_s)\end{aligned} \tag{15}$$

$$\dot{x}_5 = u \tag{16}$$

$$\Delta z_s = z_w - z_b, \quad \Delta v_s = v_w - v_b$$

$$\Delta z_t = z_r - z_w, \quad \Delta v_t = \dot{z}_r - v_w$$

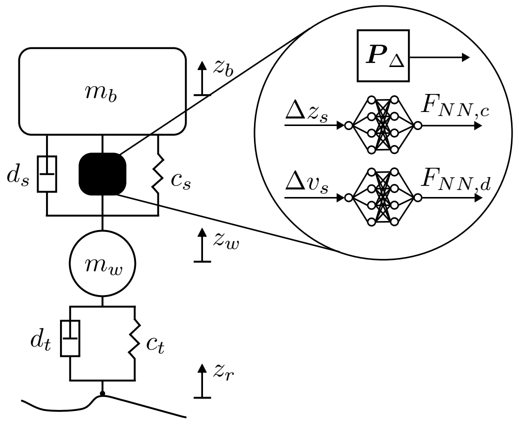


Figure 1: Scheme of the hybrid quarter vehicle model with the incorporated black-box components.

4.2 Data Generation

For the generation of training data, the linear QVM is extended by two non-linear effects in the suspension. A translational friction force F_{fr}, consisting of Stribeck-, Coulomb and viscous friction, is added (cf. Simscape: Translational Friction[4]) and the spring is modified towards a progressive characteristic by adding a quadratic term F_{pr}. The model equations (13) and (15) are consequently changed to

$$\begin{aligned}\dot{x}_2 = m_b^{-1}(&c_s\Delta z_s + d_s\Delta v_s \\ &+ F_{pr}(\Delta z_s) + F_{fr}(\Delta v_s)) \qquad (17)\\ \dot{x}_4 = m_w^{-1}(&c_t\Delta z_t + d_t\Delta v_t - c_s\Delta z_s - d_s\Delta v_s \\ &- F_{pr}(\Delta z_s) - F_{fr}(\Delta v_s))\ . \qquad (18)\end{aligned}$$

To obtain the trajectory of states and outputs, the non-linear model is simulated over time for a given input $u(t)$. The input is derived from a realistic road profile (ISO8608, Type D (Múčka, 2017)), imitating a rather rough road. The generated states and observables are sampled with 1000 Hz and are artificially disturbed by Gaussian noise. The training dataset consists of a single trajectory of 42 s length.

4.3 The Neural QVM

For the demonstration of our method, the basic linear model is augmented with two NNs to learn the missing effects from the data. The NNs are interpreted as neural compliance and -resistance (here: spring and damper) that induce the forces $F_{NN,c}$ and $F_{NN,d}$. They consequently add to the sum of forces in the given differential equations according to:

Table 1: Architecture of the deployed neural networks.

Layer	Dimensions	Activation
1	16×1	ReLU
2	16×16	ReLU
3	1×16	Identity

$$\begin{aligned}\dot{x}_2 = m_b^{-1}(&c_s\Delta z_s + d_s\Delta v_s + F_{NN,c}(\Delta z_s) \\ &+ F_{NN,d}(\Delta v_s)) \qquad (19)\\ \dot{x}_4 = m_w^{-1}(&c_t\Delta z_t + d_t\Delta v_t - c_s\Delta z_s - d_s\Delta v_s \\ &- F_{NN,c}(\Delta z_s) - F_{NN,d}(\Delta v_s))\ . \qquad (20)\end{aligned}$$

The architecture of the neural networks is summarized in Table 1. In our approach, the neural components capture single effects within a complex system. In most cases, the learned functions will be simple and small NNs with few nodes suffice. In the given setup, the two NNs only have 321 parameters each.

In a realistic setup, the accelerations a_b and a_w are comparably easy to measure. Therefore, the model output $\boldsymbol{y}$ is used for the optimization. The loss function is a normalized root mean squared error (RMSE, Eq. (21)) with N quantities (here a_b and a_w) and M data points q_{ij} with labels $\hat{q}_{ij}$. To account for different orders of magnitude of the quantities, the standard deviation σ_{q_j} is derived from the training data and used for the normalization. Further, a regularizer to enforce zero-crossing r_{zcross} (cf. Eq. (6)) with a weight factor of $\lambda = 0.01$ is added.

$$\begin{aligned}\mathcal{L} = &\sqrt{\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N}(\frac{q_{ij}-\hat{q}_{ij}}{\sigma_{q_j}})^2} \\ &+ \lambda\|[F_{NN,c}(0), F_{NN,d}(0)]^T\|_2 \qquad (21)\end{aligned}$$

When mini-batching is used, the presented loss is calculated for each sequence of the batch and averaged over all sequences.

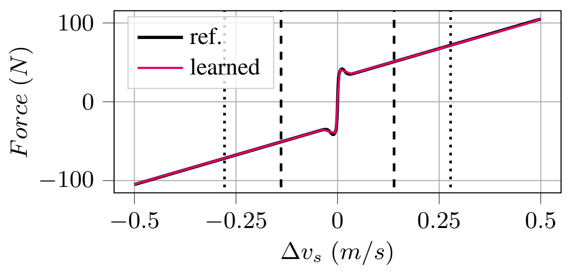
4.4 Implementation Details

The model is implemented and trained in Julia, using the packages `OrdinaryDiffEq.jl` for the solution of the ODEs, `Zygote.jl` for the automatic differentiation and tools like the ADAM optimizer from `Flux.jl`. All trainings were conducted on a notebook with a 10th generation intel i7 CPU.

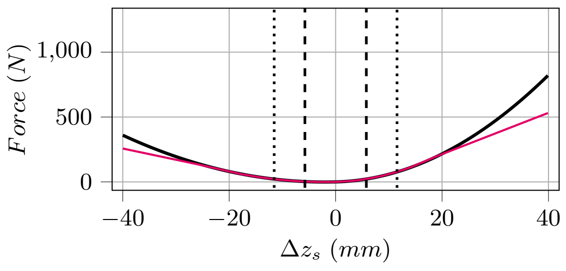
5 EXPERIMENTAL RESULTS

We use the neural QVM to prove the concept of PeN-ODEs and the sketched training methods in several

[4]https://www.mathworks.com/help/simscape/ref/translationalfriction.html, accessed 27.04.2023



(a) Friction Force



(b) Spring Characteristic

Figure 2: Comparison of the learned forces to their references. The dashed and dotted lines indicate the 1σ- and 2σ-intervals of the inputs that occurred during the training.

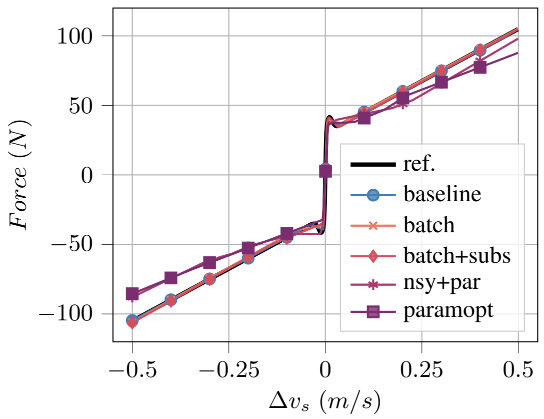
experiments. First, we show the results of a basic training and analyse them with respect to physical feasibility, interpretability and extrapolation properties. Second, the benefits of mini-batching and sub-sampling are presented. Finally, the robustness of the training against model uncertainties and noisy data is investigated and the advantages of optimizing model parameters alongside the NNs are illustrated.

5.1 Basic Training: Results and Interpretation

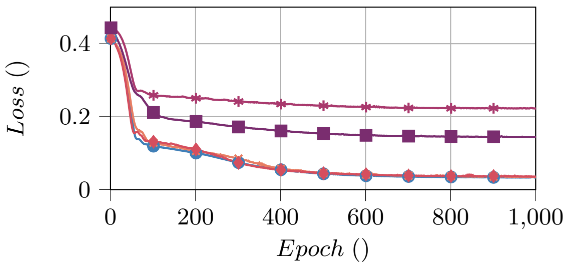
The results in this paragraph are obtained with the straightforward approach to use the whole sequence with full sample rate for each update. The training comprised 1000 epochs, i.e. simulation, loss-calculation, gradient-calculation and one update-step of the optimization. In Figure 2, the learned neural forces are compared to the non-linearities that were incorporated during the data generation. The dashed and dotted lines depict the 1σ- and 2σ-intervals (i.e. one/two standard deviations) of the NN-inputs that occurred during the training.

It shows, that the underlying non-linearities are fairly well captured by the neural components within the 2σ-confidence-interval. The outputs outside the interval are interpreted as extrapolations, since less than 10 % of the training inputs lay outside this interval. Both neural forces show a linear extrapolation behaviour that results from the ReLU activation function. Consequently, if a saturating extrapolation is preferred, the usage of a $\tanh$ activation in the last hidden layer can be considered.

We would like to emphasize that the underlying non-linearities were precisely learned from only one training sequence of 42 s length. In this case, more data would be beneficial only if the induced inputs expand the confidence intervals of the learned forces.



(a) Learned neural friction forces.



(b) Convergence of the training-loss.

Figure 3: Overview of the learned neural friction forces and the convergence of the loss for different training setups (cf. Section 5.1 - 5.4).

5.2 Mini-Batching and Sub-sampling

As described in Section 3, mini-batching and sub-sampling reduce the computational complexity of the training routine and allow the usage of larger datasets. As an indicator of the computational complexity, the

Table 2: Comparison of the training and average epoch duration.

	Baseline	**Batching**	**Batching + Sub-sampling**
Overall duration	33 h	6.5 h	4.5 h
Per epoch	120 s	24 s	17 s

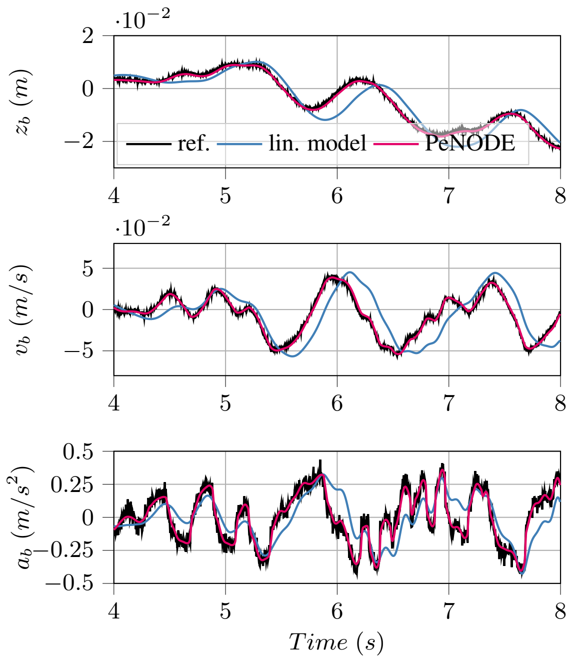


Figure 4: Detail of the model predictions for body-height, -velocity and -acceleration of the linear model compared to the trained neural ODE.

duration of one training with 1000 epochs for different setups is measured and the results are compared in Table 2. During the baseline training, each update is based on the complete sequence of 42 s and the full sample rate of 1000 Hz. In contrast, the batch updates only use a set of 3 randomly sampled sequences, each of 4 s length. With sub-sampling, the same settings (3 × 4 s) are used, but the loss is only calculated for 25 % of the available points that are also randomly sampled. The learned neural damper forces can be compared in Figure 3a (baseline, batch, batch+subs). Additionally, Figure 3b depicts the convergence of the loss over the course of the training. It shows, that the learned forces only differ slightly and the loss reaches almost the same optimum, while mini-batching decreases the duration by roughly 80 % and sub-sampling by another 30 %.

All following results were obtained with 1000 training-epochs and mini-batching as presented. Sub-sampling was not used, since the benefit is not as significant and the training becomes more prone to local optima.

5.3 Model Uncertainties and Noisy Data

The previously presented results prove the principle of learning underlying non-linearities from model output data. However, the simulated data was of high

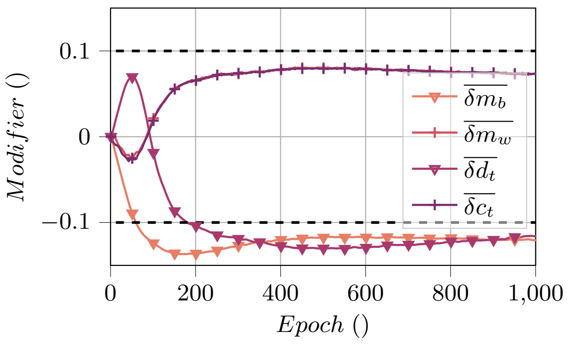


Figure 5: Convergence of the simultaneously optimized parameter modifiers.

quality and the physical part of the hybrid model used the nominal parameters. In the following, the robustness of the approach against noisy data and model uncertainties is investigated. To this end, the added noise is increased by a factor of five and the parameters of the neural QVM, i.e. the masses m_b, m_w and the coefficients of the tire spring-damper pair c_t and d_t are disturbed by $\pm$ 10 % with respect to the nominal value.

The training results under these circumstances are included in Figure 3 (nsy+par). It shows, that the neural damper still captures the non-linear friction, with a minor loss of precision. The same is true for the neural spring. Figure 4 gives an impression of the model predictions before and after the training in this case, illustrating how the learned components enhance the model predictions despite the disturbed parameters and presence of noise.

5.4 Simultaneous Parameter Fitting

Here we used the same setup as before (increased noise, disturbed parameters) but the model-parameters m_b, m_w, c_t and d_t are optimized alongside the parameters of the NNs. The corresponding modifiers are saturated with $\gamma = 0.2$ as presented in Section 2 (Eq. (7) and (8)). Figure 5 illustrates how the modifiers converge to a near optimal value over the course of the training. The target values for the modifiers marked with plus-markers are $+0.1$ and the modifiers with triangles -0.1 respectively. Figure 3, lines “nsy+par” versus “paramopt”, shows the significant decrease of the remaining loss after the optimization that can be attributed to the corrected model parameters.

6 CONCLUSIONS

In this work, we presented an approach to incorporate low-dimensional neural networks into the differential equations of a given model, forming a Physics-enhanced Neural ODE. It showed, that missing effects are effectively learned by these neural components, while physically meaningful behaviour can be enforced during the training. The usage of single-output neural networks yields the possibility to assign a confidence interval to the obtained functions, which increases the credibility of the hybrid model. We are confident, that the presented method poses a straightforward approach to enhance existing dynamical models without sacrificing their physical integrity.

Acknowledgements

This work was organized within the European ITEA3 Call6 project UPSIM - Unleash Potentials in Simulation (number 19006). The work was partially funded by the German Federal Ministry of Education and Research (BMBF, grant number 01IS20072H).

The term *Physics-enhanced Neural ODEs (PeNODE)* was created in the proposal for the ITEA 4 project OpenSCALING.

REFERENCES

Ajay, A., Wu, J., Fazeli, N., Bauza, M., Kaelbling, L. P., Tenenbaum, J. B., and Rodriguez, A. (2018). Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*.

Bruder, F. and Mikelsons, L. (2021). Modia and Julia for Grey Box Modeling. In *Linköping Electronic Conference Proceedings*. Linköping University Electronic Press.

Chang, B., Chen, M., Haber, E., and Chi, E. H. (2019). AntisymmetricRNN: A Dynamical System view on recurrent Neural Networks. *ICLR 2019*.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural Ordinary Differential Equations. *Neural Information Processing Systems*.

Daw, A., Karpatne, A., Watkins, W. D., Read, J. S., and Kumar, V. (2022). Physics-Guided Neural Networks (PGNN): An Application in Lake Temperature Modeling. In *Knowledge-Guided Machine Learning*. Chapman and Hall.

Haber, E. and Ruthotto, L. (2017). Stable Architectures for Deep Neural Networks. *Inverse Problems, Volume 34*.

Heiden, E., Millard, D., Coumans, E., Sheng, Y., and Sukhatme, G. S. (2020). NeuralSim: Augmenting Differentiable Simulators with Neural Networks. *IEEE International Conference on Robotics and Automation (ICRA) 2021*.

Hogan, N. and Breedveld, P. (2005). The Physical Basis of Analogies in Physical System Models. In *Mechatronics - An Introduction*. CRC Press.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. (2021). Physics-informed Machine Learning. *Nature Reviews Physics*.

Múčka, P. (2017). Simulated Road Profiles According to ISO 8608 in Vibration Analysis. *Journal of Testing and Evaluation*.

Owoyele, O. and Pal, P. (2022). ChemNODE: A neural Ordinary Differential Equations Framework for efficient Chemical Dinetic Solvers. *Energy and AI*.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A. (2020). Universal Differential Equations for Scientific Machine Learning.

Rai, R. and Sahu, C. K. (2020). Driven by Data or Derived through Physics? a Review of Hybrid Physics Guided Machine Learning Techniques with Cyber-Physical System (CPS) Focus. *IEEE Access*.

Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). Physics-Informed Neural Networks: A deep Learning Framework for Solving Forward and Inverse Problems involving nonlinear Partial Differential Equations. *Journal of Computational Physics*.

Ramadhan, A., Marshall, J. C., Souza, A. N., Lee, X. K., Piterbarg, U., Hillier, A., Wagner, G. L., Rackauckas, C., Hill, C., Campin, J.-M., and Ferrari, R. (2022). Capturing missing Physics in Climate Model Parameterizations using Neural Differential Equations. *Journal of Advances in Modeling Earth Systems (JAMES)*.

Thummerer, T. and Mikelsons, L. (2023). Eigen-informed NeuralODEs: Dealing with stability and convergence issues of NeuralODEs.

Thummerer, T., Stoljar, J., and Mikelsons, L. (2022). NeuralFMU: Presenting a Workflow for integrating hybrid NeuralODEs into Real-World Applications. *Electronics*.

Thummerer, T., Tintenherr, J., and Mikelsons, L. (2021). Hybrid Modeling of the Human Cardiovascular System using NeuralFMUs. *10th International Conference on Mathematical Modeling in Physical Sciences*.

Turan, E. M. and Jaschke, J. (2022). Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*.

Willard, J., Jia, X., Xu, S., Steinbach, M., and Kumar, V. (2020). Integrating Physics-Based Modeling with Machine Learning: A Survey. *ACM Computing Surveys*.

Zeng, A., Song, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2020). TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. *IEEE Transactions on Robotics*.

Zimmer, D. (2016). Equation-Based Modeling with Modelica – Principles and Future Challenges. *SNE Simulation Notes Europe*.