Task-optimal data-driven surrogate models for eNMPC via differentiable simulation and optimization

Daniel Mayfrank [*,****] **Na Young Ahn** [*]
Alexander Mitsos [***,*,**] **Manuel Dahmen** [*,†]

[*] *Forschungszentrum Jülich GmbH, Institute of Climate and Energy Systems, Energy Systems Engineering (ICE-1), Jülich 52425, Germany*
[**] *RWTH Aachen University, Process Systems Engineering (AVT.SVT), Aachen 52074, Germany*
[***] *JARA-ENERGY, Jülich 52425, Germany*
[****] *RWTH Aachen University, Aachen 52062, Germany*
[†] *Corresponding author: m.dahmen@fz-juelich.de*

Abstract: We present a method for end-to-end learning of Koopman surrogate models for optimal performance in a specific control task. In contrast to previous contributions that employ standard reinforcement learning (RL) algorithms, we use a training algorithm that exploits the potential differentiability of environments based on mechanistic simulation models to aid the policy optimization. We evaluate the performance of our method by comparing it to that of other training algorithms on an existing economic nonlinear model predictive control (eNMPC) case study of a continuous stirred-tank reactor (CSTR) model. Compared to the benchmark methods, our method produces similar economic performance while eliminating constraint violations. Thus, for this case study, our method outperforms the others and offers a promising path toward more performant controllers that employ dynamic surrogate models.

Keywords: Koopman; Reinforcement learning; Differentiable simulation; End-to-end learning; Economic model predictive control; Chemical process control

1. INTRODUCTION

Economic model predictive control (eNMPC) is a control strategy that uses a dynamic process model to predict the system behavior and make real-time control decisions by repeatedly solving an optimal control problem (OCP) in a rolling horizon fashion. Whereas traditional model predictive control (MPC) focuses on following reference trajectories, eNMPC aims to directly optimize economic process performance by integrating an economic objective into the OCP. eNMPC relies on a sufficiently accurate dynamic model of the process. Unfortunately, for high-dimensional nonlinear systems, the computational burden of solving the resulting OCPs can render eNMPC computationally intractable. In such cases, data-driven *surrogate* models for computationally expensive mechanistic dynamic models can enable real-time eNMPC by reducing the computational burden of solving the underlying OCPs (McBride and Sundmacher (2019)).

Recent articles (e.g., Gros and Zanon (2019); Mayfrank et al. (2024)) have established end-to-end reinforcement learning (RL) of dynamic surrogate models as way to train models for optimal performance in a specific (control) task (see Fig. 1a). Task-optimal dynamic models for control can be learned by viewing a dynamic model and its learnable parameters as part of a differentiable policy (see Fig. 1b). This policy consists of the dynamic model and a differentiable optimal control algorithm. Various methods for turning (e)NMPC controllers into differentiable and thereby learnable policies have been developed, see, e.g., (Amos et al. (2018); Gros and Zanon (2019); Mayfrank et al. (2024)). These methods do not depend on any specific policy optimization algorithm. End-to-end RL of surrogate models may yield increased performance of the resulting eNMPCs regarding the respective control objective, e.g., the minimization of operating costs while avoiding constraint violations (Mayfrank et al. (2024)).

RL algorithms are a class of policy optimization algorithms that enable learning of optimal controllers or dynamic models for use in eNMPC through trial-and-error actuation of real-world or simulated environments. Standard RL algorithms view environments as black boxes and do not use derivative information regarding the environment dynamics or the reward signals, even though many RL publications use simulated environments where analytical gradients of dynamics and rewards could be available. Policy gradient algorithms are the most commonly used class of RL algorithms for end-to-end learning of dynamic models for control, see, e.g., (Gros and Zanon (2019)). However, fundamental issues arise from the fact that policy gradient algorithms do *not* leverage analytical gradients

⋆ This work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centers.

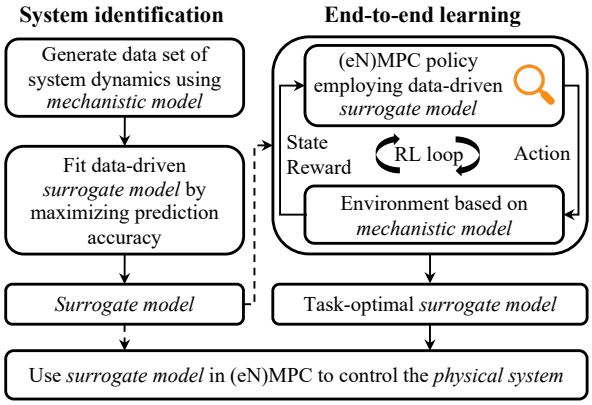

Fig. 1. (a) Comparison of two paradigms for the training of data-driven dynamic surrogate models for use in eNMPC. (b) The differentiable eNMPC policy takes as input the current state $\boldsymbol{x}_t$ and computes the optimal control action $\boldsymbol{u}_t^*$ based on a cost function f, inequality constraints $\boldsymbol{g}$, and the learnable discrete-time dynamic surrogate model $\boldsymbol{h_\theta}$ (highlighted in blue font), which is parameterized by $\boldsymbol{\theta}$.

from the environment. These issues concern both a lacking understanding of the algorithms' empirical behavior (Ilyas et al. (2018); Wu et al. (2022)) and their performance (Islam et al. (2017); Henderson et al. (2018)).

Recently, however, policy optimization algorithms that leverage the derivative information from simulated environments were designed, e.g., the Short-Horizon Actor-Critic (SHAC) algorithm (Xu et al. (2022)). These algorithms manage to avoid the well-known problems of Backpropagation Through Time (BPTT) (Werbos (1990)), i.e., noisy optimization landscapes and exploding/vanishing gradients (Xu et al. (2022)), and have shown enhanced training wall-clock time efficiency and increased terminal performance compared to state-of-the-art RL algorithms that do not exploit derivative information from the environment. These algorithmic advances could also benefit the learning of dynamic surrogate models for (eN)MPC if the mechanistic simulation model is differentiable. To this end, differentiable simulators, e.g., (Chen et al. (2018)), can be used to construct simulated RL environments with automatically differentiable dynamics and reward functions, thus enabling the use of analytic gradients for policy optimization. Nevertheless, policy optimization using differentiable environments has yet to be established for the learning of dynamic surrogate models for (eN)MPC.

By combining our previously proposed method for end-to-end learning of task-optimal Koopman models in (e)NMPC applications (Mayfrank et al. (2024)) with the SHAC algorithm (Xu et al. (2022)), we present a method for end-to-end optimization of Koopman surrogate models. Crucially, our method exploits the differentiability of simulated environments, distinguishing it from previous contributions, which are based on RL (Gros and Zanon (2019); Mayfrank et al. (2024)) or imitation learning (Amos et al. (2018)). We evaluate the resulting control performance on an eNMPC case study derived from a literature-known continuous stirred-tank reactor model (Flores-Tlacuahuac and Grossmann (2006)). We compare the performance to that of eNMPCs employing Koopman surrogate models that were trained either using (i) system identification or (ii) RL. We find that the novel combination of a Koopman-eNMPC trained using SHAC exhibits superior performance compared to the other options. This finding confirms our expectation that the advantages of policy optimization algorithms that leverage derivative information from differentiable environments can apply to the end-to-end training of dynamic surrogate models for predictive control applications. Thus, our work constitutes a step towards more performant real-time capable optimal control policies for large-scale, nonlinear systems, where optimal control policies based on a mechanistic model are not real-time capable.

We structure the remainder of this paper as follows: Section 2 presents our method. Section 3 showcases the performance of our method on a simulated case study and discusses the results. Section 4 draws some final conclusions.

2. METHOD

From a methodological perspective, the core contribution of this work is combining: (i) the SHAC algorithm (Xu et al. (2022)), a policy optimization algorithm that leverages derivative information from a differentiable simulation environment, (ii) an extension of Koopman theory to controlled systems by Korda and Mezić (Korda and Mezić (2018)) that results in convex OCPs, and (iii) our previously published (Mayfrank et al. (2024)) method for turning Koopman-(e)NMPCs into differentiable policies. We refer the reader to the aforementioned publications for a detailed description of the theoretical background of this work.

We adopt an RL perspective (Sutton and Barto (2018)) on policy optimization problems. Herein, the control problem is represented by a discrete-time Markov Decision Process (MDP) with associated states $\boldsymbol{x}_t \in \mathbb{R}^n$ and control inputs $\boldsymbol{u}_t \in \mathbb{R}^m$, a transition function $\boldsymbol{\mathcal{F}} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, $\boldsymbol{x}_{t+1} = \boldsymbol{\mathcal{F}}(\boldsymbol{x}_t, \boldsymbol{u}_t)$, and a scalar reward function $\mathcal{R} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $r_{t+1} = \mathcal{R}(\boldsymbol{x}_{t+1}, \boldsymbol{u}_t)$. An *environment* is the MDP of a specific RL problem. An *episode* refers to a sequence of interactions between a policy and its environment, starting from an initial state, involving a series of control inputs, and leading to a terminal state. A policy $\boldsymbol{\pi_\theta}(\boldsymbol{u}_t|\boldsymbol{x}_t) : \mathbb{R}^n \to \mathbb{R}^m$ is a function, parameterized by $\boldsymbol{\theta}$, mapping states $\boldsymbol{x}_t$ to (probability distributions over) control inputs $\boldsymbol{u}_t$. The goal of policy optimization is to maximize the expected future sum of rewards.

We aim to exploit the differentiability of continuous-time mechanistic models that can be represented as ordinary differential equation (ODE) systems, i.e.,

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

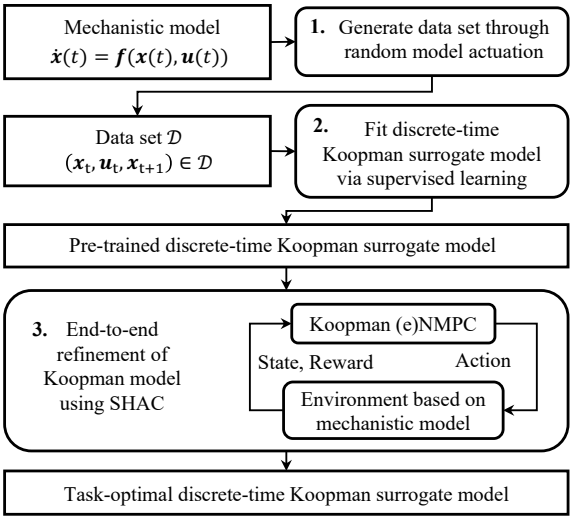

Fig. 2. Workflow from mechanistic model to task-optimal dynamic Koopman surrogate model. Adapted from Mayfrank et al. (2024).

for the end-to-end learning of task-optimal discrete-time dynamic surrogate models. In our previous publication (Mayfrank et al. (2024)), we present a method for end-to-end RL of data-driven Koopman models for optimal performance in (e)NMPC applications, based on viewing the predictive controller as a differentiable policy and training it using RL. This method is independent of the specific policy optimization algorithm. Therefore, by replacing the RL algorithm (Schulman et al. (2017)) with SHAC (Xu et al. (2022)), we can take advantage of policy optimization algorithms that exploit the differentiability of simulated environments in the learning of surrogate models for dynamic optimization. Analogous to the approach we take in (Mayfrank et al. (2024)), the overall workflow (visualized in Fig. 2) from a mechanistic model to a task-optimal dynamic Koopman surrogate model consists of three steps: (*i*) We generate a data set of the system dynamics by simulating the mechanistic model using randomly generated control inputs. (*ii*) Following the model structure proposed by Korda and Mezić (2018), we fit a Koopman model (Koopman (1931)) with learnable parameters $\boldsymbol{\theta}$ to the data. (*iii*) Using the mechanistic process model (Eq. 1) and a differentiable simulator (Chen et al. (2018)), we construct a differentiable RL environment whose reward formulation incentivizes task-optimal controller performance on a specific control task. For instance, in an eNMPC application, the task-specific reward should depend on operating costs and potential constraint violations, not on the prediction accuracy of the dynamic model, which is used as part of the predictive controller. Using the differentiable environment, we fine-tune the Koopman model for task-optimal performance as part of a predictive controller. Fig. 3 provides a conceptual sketch of this fine-tuning process. To ensure exploration in the training process, we add Gaussian noise to the – otherwise deterministic – output of the Koopman eNMPC policy. For a detailed description of steps one and two in Fig. 2 (data generation and SI) and how to construct a differentiable eNMPC policy from a Koopman surrogate model, we refer the reader to our previous work (Mayfrank et al. (2024)).

3. NUMERICAL EXPERIMENTS

3.1 Case study description

Following our previous work (Mayfrank et al. (2024)), we consider a dimensionless benchmark continuous stirred-tank reactor (CSTR) model (Flores-Tlacuahuac and Grossmann (2006)) that consists of two states (product concentration c and temperature T), two control inputs (production rate ρ and coolant flow rate F), and two nonlinear ordinary differential equations:

$$\dot{c}(t) = (1 - c(t))\frac{\rho(t)}{V} - c(t)ke^{-\frac{N}{T(t)}},$$
$$\dot{T}(t) = (T_f - T(t))\frac{\rho(t)}{V} + c(t)ke^{-\frac{N}{T(t)}} - F(t)\alpha_c(T(t) - T_c)$$

The model has a steady state at $c_{\mathrm{ss}} = 0.1367$, $T_{\mathrm{ss}} = 0.7293$, $\rho_{\mathrm{ss}} = 1.0\frac{1}{\mathrm{h}}$, $F_{\mathrm{ss}} = 390.0\frac{1}{\mathrm{h}}$. Based on the model, we construct an eNMPC application by assuming that the electric power consumption is proportional to the coolant flow rate F, enabling production cost savings by shifting process cooling to intervals with comparatively low electricity prices. Given price predictions, the goal is to minimize the operating costs while adhering to process constraints. The state variables and the control inputs are subject to box constraints ($0.9c_{\mathrm{ss}} \leq c \leq 1.1c_{\mathrm{ss}}$, $0.8T_{\mathrm{ss}} \leq T \leq 1.2T_{\mathrm{ss}}$, $0.8\frac{1}{\mathrm{h}} \leq \rho \leq 1.2\frac{1}{\mathrm{h}}$, and $0.0\frac{1}{\mathrm{h}} \leq F \leq 700.0\frac{1}{\mathrm{h}}$). We include a product storage with a maximum capacity of six hours of steady-state production to enable flexible production. To match the hourly structure of the day-ahead electricity market, we choose control steps of length $\Delta t_{\mathrm{ctrl}} = 60$ minutes. A more detailed case study description, including the model parameters, is given in Mayfrank et al. (2024).

3.2 Training setup

We compare the performance of the following three training paradigms:

(1) *Koopman-SI*: eNMPC controller using a Koopman surrogate model trained solely using SI.
(2) *Koopman-PPO*: eNMPC controller using a Koopman surrogate model pretrained using SI and refined for task-optimal performance using the state-of-the-art policy gradient algorithm Proximal Policy Optimization (PPO) (Schulman et al. (2017)), like we did in Mayfrank et al. (2024).
(3) *Koopman-SHAC* (main contribution of this work): eNMPC controller using a Koopman surrogate model pretrained using SI and refined for task-optimal performance using the SHAC algorithm (Xu et al. (2022)).

Our goal is to train dynamic surrogate models of fixed size for optimal performance as part of eNMPC. All results presented in Subsection 3.3 are obtained using a Koopman model with a latent space dimensionality of eight, i.e., $\boldsymbol{A_\theta} \in \mathbb{R}^{8\times 8}, \boldsymbol{B_\theta} \in \mathbb{R}^{8\times 2}, \boldsymbol{C_\theta} \in \mathbb{R}^{2\times 8}$, and an encoder $\boldsymbol{\psi}\colon \mathbb{R}^2 \to \mathbb{R}^8$ in the form of a feedforward neural network with two hidden layers, four and six neurons, respectively and hyperbolic tangent activation functions. We use an eNMPC horizon of nine hours.

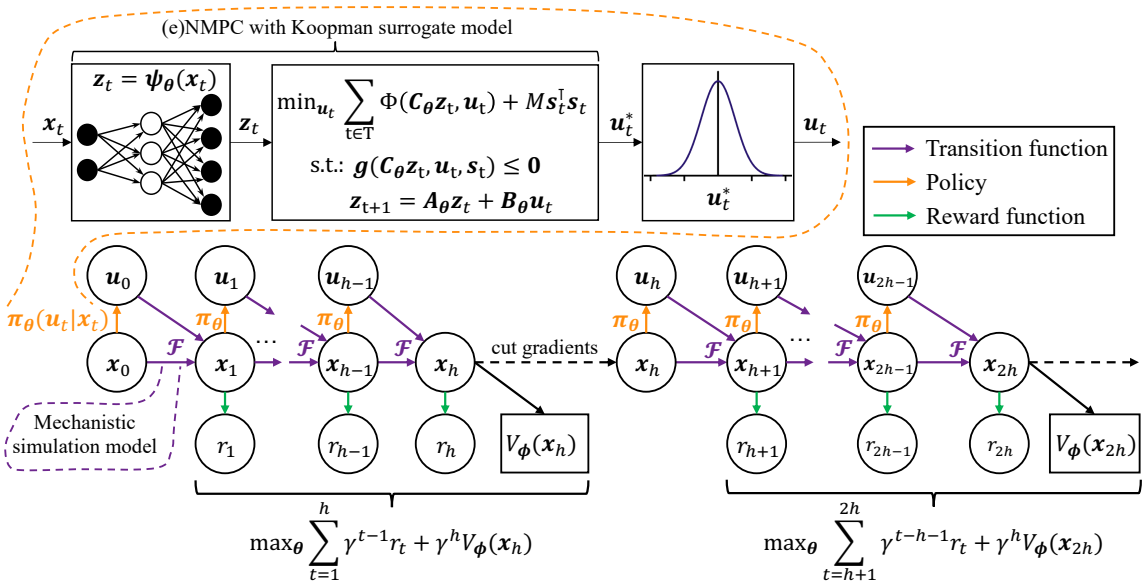

Fig. 3. Using SHAC to train a task-optimal Koopman surrogate model for the transition function $\mathcal{F}$. This figure can be interpreted as a SHAC-specific unrolled version of the typical RL loop shown in the third step in Fig. 2. The policy is optimized by adjusting the parameters $\boldsymbol{\theta}$ of the dynamic Koopman surrogate model. Φ is a convex function for the stage cost of the objective function. To ensure the feasibility of the resulting optimal control problems, we add slack variables $\boldsymbol{s}_t$ to the state bounds (Mayfrank et al. (2024)). Their use is penalized quadratically using a penalty factor M. Due to the use of PyTorch and *cvxpylayers* (Agrawal et al. (2019)), the output $\boldsymbol{u}_t$ of the policy is differentiable with respect to $\boldsymbol{x}_t$ and $\boldsymbol{\theta}$. The critic is a feedforward neural network with trainable parameters $\boldsymbol{\phi}$. To increase the clarity of the figure, we omit the direct dependence of r_{t+1} with respect to $\boldsymbol{u}_t$.

We use the same data set as in (Mayfrank et al. (2024)) for the SI pretraining of the Koopman surrogate model. This data set consists of 84 trajectories, each of a length of 5 days, i.e., 480 time steps, using a step length of 15 minutes. Of those 84 trajectories, we use 63 for training and the remaining 21 for validation. Then, we perform SI of the Koopman model in a similar way as described in (Mayfrank et al. (2024)). We repeat SI ten times using random seeds. We use the model with the lowest validation loss for the *Koopman-SI* controller. The same model is used as the initial guess when training the *Koopman-PPO* and *Koopman-SHAC* controllers.

In order to use a policy optimization algorithm such as SHAC, which makes use of derivative information from a simulated environment, not only the dynamic model but also the reward function must be differentiable. For our case study, we choose a reward function that calculates the reward at time step t based on whether any constraints were violated at that time step, and on the electricity cost savings compared to the steady-state production at nominal rate between $t-1$ and t. The constraint component r_t^{con} of the reward quadratically penalizes violations of the bounds of c, T, and the product storage, i.e., $r_t^{\text{con}} \geq 0$, and $r_t^{\text{con}} = 0$ if no constraint violation occurs at t. The cost-component r_t^{cost} of the reward is calculated by taking the difference between the cost at nominal production and the actual production cost between $t-1$ and t, i.e.,

$$r_t^{\text{cost}} = (F_{\text{ss}} - F_{t-1}) \cdot p_{t-1} \cdot \Delta t_{\text{ctrl}},$$

where p_{t-1} is the electricity price between $t-1$ and t, and Δt_{ctrl} is the time between $t-1$ and t for which the controls are held constant. We balance the influence of the two components on the overall reward using a hyperparameter α:

$$r_t = \alpha \cdot r_t^{\text{cost}} - r_t^{\text{con}}$$

We train the policies using day-ahead electricity prices from the Austrian market from March 29, 2015, to March 25, 2018 (Open Power System Data (2020)). Using random seeds, we repeat the controller training ten times for each combination of policy and training algorithm (except for the *Koopman-SI* controller, whose Koopman model is not trained any further after SI).

We use the same hyperparameters for *Koopman-PPO* and *Koopman-SHAC* wherever possible, i.e., for all hyperparameters which are not part of PPO or SHAC. For the hyperparameters of PPO and SHAC, we do not perform extensive hyperparameter tuning. Instead, we mostly rely on the standard values. In both algorithms, we use the Adam optimizer with a small learning rate of 10^{-5} as the behavior of the Koopman eNMPCs is highly sensitive to small changes in the parameters. We used the *Stable-Baselines3* (Raffin et al. (2021)) implementation of PPO but implemented our own version of SHAC following the description in (Xu et al. (2022)). All code used for training the controllers, including the hyperparameters that were used to obtain the results presented in Subsection 3.3, is publicly available[1]. In addition to the code used to obtain the results presented in the following subsection, the linked repository also contains code for training pure neural network policies for our case study using PPO and SHAC. As the results of the neural network policies do not

[1] `https://jugit.fz-juelich.de/iek-10/public/optimization/shac4koopmanenmpc`

influence the narrative of this work in a meaningful way we do not discuss these results.

3.3 Results

For *Koopman-PPO* and *Koopman-SHAC*, we identify the controller (and the associated set of parameters) that achieved the highest average reward between two consecutive parameter updates. We test their performance and that of the *Koopman-SI* controller on a continuous roughly half-year-long test episode using Austrian day ahead electricity price data from March 26, 2018, to September 30, 2018 (Open Power System Data (2020)). Unlike the training process, we perform this testing without exploration, i.e., we waive adding Gaussian noise to the controller output (cf. Fig. 3). We initialize the test episode for each controller at the steady state of the CSTR and with empty product storage. The results are presented in Table 1. The trajectories produced by all controllers exhibit an intuitive inverse relationship between electricity prices and coolant flow rate.

Table 1. Test results: The economic cost is stated relative to the nominal production cost, i.e., we report the cost incurred by the respective controller, divided by the cost incurred by steady-state production at nominal rate given the same electricity price trajectory. The percentage of control steps that result in constraint violations is given. The average size of a constraint violation is given relative to the size of the feasible range of the variable whose bound was violated.

	Economic cost	Constr. viols. [%]	Avg. size constr. viol.
Koopman-SI	**0.88**	19.88	$5.1 \cdot 10^{-2}$
Koopman-PPO	0.90	17.57	$1.3 \cdot 10^{-2}$
Koopman-SHAC	0.90	**0.0**	–

As can be seen from Table 1, *Koopman-SHAC* is the only controller that does not produce any constraint violations. *Koopman-SI* achieves slightly higher cost savings than *Koopman-SHAC*, however, it frequently produces constraint violations. Therefore, we consider *Koopman-SHAC* preferable in most real-world applications where constraint-satisfaction is of high importance.

It is noticeable that *Koopman-SI* and *Koopman-PPO* cause substantially more constraint violations than the *Koopman-SHAC* controller. In the case of *Koopman-SI*, this is not surprising since the employed Koopman surrogate model received no end-to-end training for task-optimal performance. Here, frequent but minor constraint violations show that the controller is trying to operate at the borders of the feasible range, which is normal behavior for a predictive controller. However, in the case of *Koopman-PPO* the results are unsatisfactory and rather unexpected. Here, the end-to-end training reduced the average size of the constraint violations by roughly a factor of four, but only led to a small improvement in the frequency of constraint violations. We observe that some training runs did not improve performance compared to *Koopman-SI* at all. Moreover, those training runs that did improve performance did not show stable

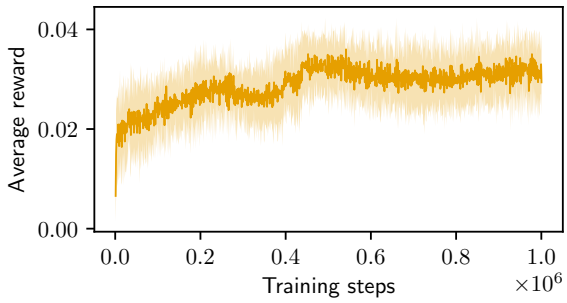

Fig. 4. Learning progress in the *Koopman-SHAC* training runs. The dark orange line indicates the running average reward over the previous 1024 steps in the environment, averaged over all ten training runs. The light orange region indicates one standard deviation of the performance variance between the training runs.

convergence to high rewards. The unstable convergence of the *Koopman-PPO* controllers is in line with the results in our previous work (Mayfrank et al. (2024)). There, we used a non-differentiable reward formulation with a high constant penalty incurred by any constraint violation. Using that reward formulation, we managed to substantially reduce the frequency of constraint violations unlike here. However, that approach (Mayfrank et al. (2024)) severely affected the resulting economic performance and thus produced overly conservative eNMPC controllers. In contrast to *Koopman-PPO*, *Koopman-SHAC* exhibits a relatively stable convergence to high rewards in our case study (Fig. 4) and produces superior terminal performance (Tab. 1). The control performance improves relatively evenly in all ten training runs without ever dropping for extended periods.

Due to the small size of the case study under consideration, a detailed analysis of the training runtimes would provide little insight about the expected runtime on a control problem of more practical relevance. Therefore, we leave such an analysis for future work on larger systems.

Policy gradient analysis The fundamental difference between the two policy optimization algorithms SHAC (Xu et al. (2022)) and PPO (Schulman et al. (2017)) is that SHAC utilizes analytic policy gradients from an automatically differentiable environment, whereas PPO estimates policy gradients via the policy gradient theorem (Sutton and Barto (2018)). Ilyas et al. (2018) show that given common and practical hyperparameter configurations, the PPO-estimated policy gradients can incur a high variance which can lead to unstable training convergence, as we observe in our case. We follow the approach by Ilyas et al. (2018) to analyze empirically whether there is a meaningful difference in the variance of the policy gradients produced by PPO and SHAC when applied to our case study. To this end, we investigate how similar the direction of multiple policy gradients are given a fixed policy parameterization. We fix the policy parameters to that of the *Koopman-SI* controller for this analysis. Then, for PPO and SHAC, we fit the critic to this policy *without* updating the policy. Finally, for both algorithms and still without updating the policy, we record the policy gradients of 100 optimization steps. We calculate how similar the 100

recorded gradients are by computing their average pairwise cosine similarity. The cosine similarity is a measure of the similarity of two vectors which only depends on their direction, not on their length. It takes a value of one if both vectors point in the same direction, zero if they are orthogonal to each other, and minus one if they point in exactly opposite directions. PPO produces an average cosine similarity of 0.22, whereas SHAC results in an average cosine similarity of 0.94. Thus, the variance in the direction of policy gradients is much higher for PPO than for SHAC, which might explain the instable convergence to high rewards and thus the relatively bad performance of *Koopman-PPO* (see Tab. 1).

4. CONCLUSION

We combine our previously published method for turning Koopman-(e)NMPC controllers into differentiable policies (Mayfrank et al. (2024)) with the policy optimization algorithm SHAC (Xu et al. (2022)). Our approach leverages derivative information from automatically differentiable simulated environments, differentiating it from previously published methods for end-to-end training of dynamic models for control. We find that SHAC produces a stable convergence to high control performance across all independent training instances, translating to superior control performance compared to our previously published approach (Mayfrank et al. (2024)) which was based on the PPO algorithm. Note that even though our method achieves perfect constraint satisfaction in our case study, full adherence to constraints can in general not be expected.

The results can be understood as a successful proof of concept. By training data-driven surrogate models for optimal performance in a specific control task, our method utilizes the representational capacity of the model efficiently, thus avoiding unnecessarily large and computationally expensive surrogate models. We view our approach as a promising avenue toward more performant real-time capable data-driven (e)NMPCs. Still, the computational burden of backpropagation through mechanistic simulations and optimal control problems and thus the cost of each training iteration is naturally linked to the size of the mechanistic simulation model, meaning that our method could become computationally intractable for very large models. Thus, future work should investigate the application of our method to larger mechanistic simulation models and more challenging control problems, presumably necessitating training for more iterations.

ACKNOWLEDGEMENTS

We thank Jan C. Schulze (Process Systems Engineering (AVT.SVT), RWTH Aachen University, 52074 Aachen, Germany) for fruitful discussions and valuable feedback.

REFERENCES

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J.Z. (2019). Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32, 9558–9570.

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J.Z. (2018). Differentiable MPC for end-to-end planning and control. *Advances in Neural Information Processing Systems*, 31, 8299–8310.

Chen, R.T.Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 6572–6583.

Flores-Tlacuahuac, A. and Grossmann, I.E. (2006). Simultaneous cyclic scheduling and control of a multiproduct CSTR. *Industrial & Engineering Chemistry Research*, 45(20), 6698–6712.

Gros, S. and Zanon, M. (2019). Data-driven economic NMPC using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2), 636–648.

Henderson, P., Romoff, J., and Pineau, J. (2018). Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *arXiv preprint arXiv:1810.02525*.

Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2018). A closer look at deep policy gradients. *arXiv preprint arXiv:1811.02553*.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. (2017). Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*.

Koopman, B.O. (1931). Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5), 315–318.

Korda, M. and Mezić, I. (2018). Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93, 149–160.

Mayfrank, D., Mitsos, A., and Dahmen, M. (2024). End-to-end reinforcement learning of Koopman models for economic nonlinear model predictive control. *Computers & Chemical Engineering*, 190, 108824.

McBride, K. and Sundmacher, K. (2019). Overview of surrogate modeling in chemical process engineering. *Chemie Ingenieur Technik*, 91(3), 228–239.

Open Power System Data (2020). Open power system data. `https://data.open-power-system-data.org/time_series/` (accessed on 2022-08-29).

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268), 1–8.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

Werbos, P.J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10), 1550–1560.

Wu, S., Shi, L., Wang, J., and Tian, G. (2022). Understanding policy gradient algorithms: A sensitivity-based approach. In *International Conference on Machine Learning*, 24131–24149. PMLR.

Xu, J., Makoviychuk, V., Narang, Y., Ramos, F., Matusik, W., Garg, A., and Macklin, M. (2022). Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*.