Physics-Informed Neural Networks with Skip Connections for Modeling and Control of Gas-Lifted Oil Wells

Jonas Ekeland Kittelsen[a], Eric Aislan Antonelo[b], Eduardo Camponogara[b], Lars Struen Imsland[a]

[a]*Norwegian University of Science and Technology, Trondheim, Norway*
[b]*Federal University of Santa Catarina, Florianopolis, Brazil*

Abstract

Neural networks, while powerful, often lack interpretability. Physics-Informed Neural Networks (PINNs) address this limitation by incorporating physics laws into the loss function, making them applicable to solving Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs). The recently introduced PINC framework extends PINNs to control applications, allowing for open-ended long-range prediction and control of dynamic systems. In this work, we enhance PINC for modeling highly nonlinear systems such as gas-lifted oil wells. By introducing skip connections in the PINC network and refining certain terms in the ODE, we achieve more accurate gradients during training, resulting in an effective modeling process for the oil well system. Our proposed improved PINC demonstrates superior performance, reducing the validation prediction error by an average of 67% in the oil well application and significantly enhancing gradient flow through the network layers, increasing its magnitude by four orders of magnitude compared to the original PINC. Furthermore, experiments showcase the efficacy of Model Predictive Control (MPC) in regulating the bottom-hole pressure of the oil well using the improved PINC model, even in the presence of noisy measurements.

Keywords: physics-informed neural networks, gas-lifted oil well, skip connections, hierarchical architecture

1. Introduction

Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019) have been widely used to solve Partial Differential Equations (PDEs) and Ordinary Differential Equations (ODEs), representing an alternative to numerical methods that can speed up simulation orders of magnitude. Once trained, these networks can output directly the solution without access to the PDE or ODE equations. Actually, these equations are employed only during the training phase by designing a loss function that considers a measure of the deviation of the derivative of the network's output from the physics laws given by ODEs/PDEs. In this way, the PINN is trained so that its output satisfies the constraints of the physics, in addition to the traditional data loss for regression, *i.e.*, the mean squared error of the data points. Regarding an ODE or PDE, this data loss corresponds to the error concerning the initial or boundary conditions. For industrial plants represented by ODEs, one could collect more data points in addition to initial conditions to complement differential equations with uncertain or unknown parameters (Raissi et al., 2019).

Applications of PINNs are widespread in many engineering areas. Ouyang et al. (2023) employed multiple serial PINNs to decouple the governing physics equations for learning the reconstruction of hydrofoil cavitation flow. Nazari et al. (2023) successfully applied PINNs for modeling river channels and parameter discovery under conditions of limited system measurements and Wu et al. (2022) employed PINNs for fluid flow generation. Many extensions of PINNs exist. For instance, Dwivedi and Srinivasan (2020) extended Extreme Learning Machines as physics-informed networks for rapidly learning solutions to PDEs; Xiang et al. (2022) introduced an improved PINN with a self-adaptive loss function to dynamically weigh the loss terms relating to data and physics; Tang et al. (2023) devised a parallel physics-informed neural network to solve the multi-body dynamic equations in full-scale train collision simulation, while Zhou et al. (2022) developed a physics-informed generative adversarial network-based approach to facilitate uncertainty quantification and propagation in addition to enabling measurement data fusion into system reliability assessment.

PINNs have been extended for long-range simulation and control applications in Antonelo et al. (2024), as they cannot inherently cope with changing inputs or extrapolate the predictions for time periods longer than the one defined during training. To address these limitations, Antonelo et al. (2024)

2

presented a new framework called Physics-Informed Neural Nets for Control (PINC), which is a novel PINN-based architecture suitable for control problems and able to simulate for longer-range time horizons that are not fixed beforehand during training. As a result, PINC is more flexible than traditional PINNs and faster than numerical methods as it relies only on signal propagation through the network, making it less computationally costly and better suited for model simulation in Model Predictive Control, for instance. Model Predictive Control (MPC) is a widely applied method for multivariate control in both industry and academia (Camacho and Bordons, 2013), which has been used successfully in various fields such as oil and gas (Jordanou et al., 2022), aerospace (Eren et al., 2017), process industries, and robotics (Nascimento et al., 2018) since its inception in the 1970s.

The class of systems that Antonelo et al. (2024) showed PINC to be effective include the Van der Pol oscillator and the four-tank system. Despite being benchmark systems for nonlinear dynamics, their dynamic equations are smooth functions that may lack specific characteristics of real-world complex systems. These particular features, such as nonsmoothness, may hinder the training of PINC networks. Here, we investigate improved versions of the PINC framework for a class of systems that exhibit highly complex and nonsmooth behavior. In particular, we are interested in modeling and control of gas-lifted oil wells, which are discussed in Section 3. The baseline version of PINC as originally proposed did not succeed in modeling and controlling the oil wells considered in this paper. The three main reasons for that are: 1) the magnitude of the gradient of the physics-loss function was too low for training to progress consistently; 2) some ODE equations of the well model include functions not defined for negative numbers; 3) and highly nonlinear or nonsmooth terms of the ODE functions can negatively influence the training of the PINC.

For such systems, nonsmoothness impairs the learning efficiency as some terms in the ODE and, consequently, in the PINN's loss function, lack a continuous derivative at specific points within its domain, exemplified when the left-hand and right-hand derivatives at a point are unequal or do not exist (Clarke, 1990). This property captures abrupt changes in the function's behavior, such as jumps or kinks that arise in the oil well model through the maximum operator.

The complexity of the class of systems we consider is related not only to their nonsmothness, but also to: 1) the *size* of the model in terms of the number of variables and constraints in the DAE/ODE, namely the oil well

3

model has 3 state equations and 34 algebraic equations, which is far more complex than the applications in Antonelo et al. (2021) with only either two or four states, without algebraic equations and variables; 2) the nonlinear mappings between the variables, e.g., fractional exponents, ratios, and maximum operators; 3) functions in the ODE that are not defined for negative numbers, e.g., square root and logarithm. The more these occur, arguably the more complex the system is, increasing the occurrence of failure modes in PINNs (Fuks and Tchelepi, 2020; Raissi, 2018; Wang et al., 2022). Besides, some of these systems may exhibit stiffness due to the interaction of multiple physical processes that operate at different time scales.

The contributions of this work are discussed next. The first issue presented above was mainly tackled by using skip connections as in Wang et al. (2021), aiming to improve the magnitude of the gradient through the PINN's layers. In that work, they showed that traditional PINNs have specific gradient pathologies, which can be dealt with by the novel architecture and a learning rate annealing algorithm. In the current work, we transfer their proposed novel PINN architecture with skip connections to the PINC framework, showing that this improvement was essential for the modeling and controlling more complex systems such as the oil well. The remaining issues, which refer to the terms in the ODE equations, are dealt with by modifying these equations when they appear in the physics-loss function for training the PINC. This is to say that surrogate terms can replace specific terms to render training smoother and more efficient. Notice that the latter procedure is specific to the application at hand, *i.e.*, the oil well model. Lastly, we propose a hierarchical architecture with two modules: the first one corresponds to a PINC trained to predict the states of the oil well; and the second one is a standard feedforward neural network (NN) trained to predict the algebraic variables of the system (*e.g.*, bottom-hole pressure) based on the predictions of the states by the PINC (first module), *i.e.*, the output of the PINC is the input to the second NN. This NN streamlines the training of additional algebraic variables without retraining the PINC, which would be far more computationally costly than just training an NN.

The remainder of this work is organized as follows. Section 2 discusses some works from the literature related to applying the PINC framework in model predictive control. Section 3 introduces gas-lifting in oil wells and presents the differential algebraic equations that model the process. The PINN and PINC frameworks, along with the proposed improvements to the PINC, evaluation metrics, and MPC, are presented in Section 4. In the

4

following section, the experimental setup is described, while in Section 6, the results are showcased. Section 7 concludes this work.

2. Related Works

Nicodemus et al. (2022) used the PINC framework, without skip connections (unlike our work), for model predictive control of multi-body dynamics. The results show that the PINC framework effectively solves a tracking problem for a complex mechanical system, a multi-link manipulator. Applying to a PowerCube serial robot, they demonstrate that the simulation of the nonlinear dynamics with PINC speeds up the computation time relative to numerical methods while retaining sufficient accuracy.

Liu and Wang (2021) proposed a model-based Reinforcement Learning (RL) algorithm that utilizes physical laws to learn the state transition dynamics of an agent's environment. The algorithm employs an encoder-decoder recurrent network architecture to model the state transition function by minimizing the violation of conservation laws. The learned transition model is then employed to generate samples for an alternative replay buffer, which improves sample efficiency in the RL update process and reduces the need for real-world interactions. However, this approach differs from our PINC-based approach. Liu and Wang (2021) build the physics-loss function on the discretized form of the laws of the system, unlike our work that relies on the continuous form. While they use recurrent networks, our approach employs feedforward networks in which time is explicitly given as input.

Gokhale et al. (2022) applied a PINN approach to learning a control-oriented thermal model of a building. The authors assume that the model is a discrete-time transition function in a Markov Decision Process (MDP) that predicts the next state, given the current state and action. This approach allows for control actions to be input to the model. However, unlike in our approach, their physics loss function also needs to be discretized. Although their proposal is control-oriented, they do not demonstrate actual control experiments with the trained PINNs, unlike the results from our work presented in Section 6.

3. Gas-lifted Oil Wells

Primary recovery is the first stage of extracting oil and gas from reservoirs. It relies on the natural difference in pressure between the surface and

the underground reservoir, therefore requiring relatively limited capital investment. With the gradual extraction of oil from the well, the pressure underground will slowly decrease, causing the volume of oil production to decline. Secondary recovery techniques are employed to mitigate the pressure loss, such as water injection, which seeks to force oil to the surface by directly applying pressure. Also, artificial lifting techniques are applied in wells, particularly those found in deep-water offshore reservoirs, to increase the pressure gradient from the bottom hole to the surface. A widely used technique for artificial lifting is gas-lift. It works by injecting high-pressure gas at the bottom of the well, reducing the density of the production stream and forcing the flow of fluids to the surface. Gas-lift is a favored technique for its desirable features, which include low installation and maintenance cost, robustness, and a wide range of operating conditions.

The dynamic model for gas-lifted oil wells considered here is a system of Differential Algebraic Equations (DAE) developed by Jahanshahi et al. (2012). The derivation of the model simplified some algebraic equations to avoid implicit terms, resulting in an explicit model. Consequently, the DAE system can be regarded as a system of Ordinary Differential Equations (ODE) and simulated using, for instance, the Runge-Kutta method. These simplifications are imposed on the equations for velocities and will be explained further in the text.

3.1. Well Model

The model has several variables, several of which are similar in nature (pressures and densities, e.g.) but for different well locations. Subscript abbreviations are used extensively in the variables to refer to the phase (gas or liquid) and location. The notation is presented in Table 1. An important remark is that this model does not distinguish between water and oil, considering only a liquid phase.

3.1.1. States

The system state is characterized by the mass of gas in the annulus $m_{\mathtt{G,an}}$, mass of gas in the tubing $m_{\mathtt{G,tb}}$, and mass of liquid in the tubing $m_{\mathtt{L,tb}}$. The differential equations are:

$$\dot{m}_{\mathtt{G,an}} = w_{\mathtt{G,in}} - w_{\mathtt{G,inj}} \tag{1a}$$

$$\dot{m}_{\mathtt{G,tb}} = w_{\mathtt{G,inj}} + w_{\mathtt{G,res}} - w_{\mathtt{G,out}} \tag{1b}$$

$$\dot{m}_{\mathtt{L,tb}} = w_{\mathtt{L,res}} - w_{\mathtt{L,out}} \tag{1c}$$
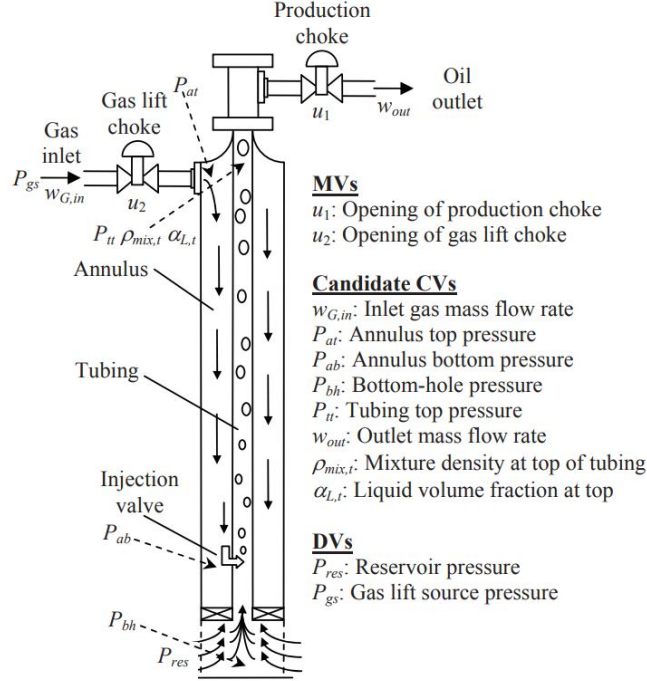
6

Figure 1: Schematics of a gas-lifted oil well ([Jahanshahi et al., 2012](#)). Liquid and gas from the reservoir enter the well at the bottom of the tubing, referred to as the bottom hole. From there, the gas and liquid flow up through the tubing until they exit through the production choke. The tubing is encased by a larger tube called the annulus. At the top of the annulus, gas is injected through the gas-lift choke. This gas flows down the annulus and enters the production tubing through a check valve (directional/one-way valve) near the annulus's bottom. This gas helps "lift" the reservoir flow up the tubing, increasing the production of the well. This kind of artificial lift is commonly used when the pressure in the reservoir is insufficient to sustain the flow from the reservoir to the top side.

where: $w_{\texttt{G,in}}$ is the gas mass flow injected into the top of the annulus through the gas-lift choke; $w_{\texttt{G,inj}}$ is the gas mass flow from the annulus into the tubing through the injection check valve, located close to the bottom of the well; $w_{\texttt{G,res}}$ and $w_{\texttt{L,res}}$ are the gas and liquid mass flows from the reservoir into the tubing; $w_{\texttt{G,out}}$ and $w_{\texttt{L,out}}$ are the gas and liquid mass flows out of the well through the production choke, or simply the gas and liquid production of the well.

The equations that calculate the flows appearing in state equations as well as the parameters for three oil well models used in the experiments are introduced in [Appendix A](#). Refer to [Jahanshahi et al. (2012)](#) for a full account

7

Table 1: Abbreviations used in the subscript of the model variables indicate the phase and location the respective variable represents.

Abbreviation	Description
G	gas
L	liquid
an	annulus
tb	tubing
bh	bottom-hole
t	top
b	bottom

of the model. The parameters appearing in the equations are explained in Table A.4.

4. Methods

4.1. Physics-Informed Neural Networks (PINN)

Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019) are trained to reproduce the solution of a dynamic system by utilizing the known (or partially known) underlying physics of the system. The use of physics knowledge greatly reduces the need for training data, in some cases removing it completely. Here, the output of the PINN represents the solution of an ODE. Let us consider the Initial Value Problem (IVP):

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0 \tag{2}$$

where we are interested in obtaining the solution $\mathbf{y}(t)$ on some interval $t \in [0, T]$. For some $\mathbf{f}(\mathbf{y})$, the IVP can be solved analytically, while for others, obtaining an analytic solution is not possible or practical, in which case we have the choice of using numerical methods, *e.g.*, Runge-Kutta method. PINNs offer an alternative way of solving such problems.

Consider the setup illustrated in Figure 2, where a neural network, with time t as input, is trained to predict the corresponding states $\mathbf{y}(t)$ as output. The network's training and prediction considers that t is within the time horizon $t \in [0, T]$. In this conventional architecture, initial conditions are "trained" into the neural network weights, which means that the trained network will work only with the specified fixed initial conditions. Thus, we would have to train a new neural network if we want to solve an IVP for the same system but with a different initial condition $\mathbf{y}(0)$.
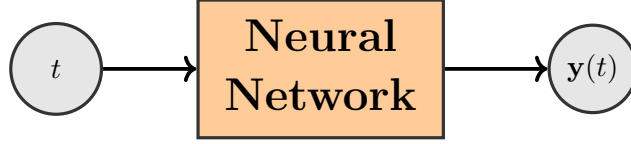
8

Figure 2: PINN structure for solving an IVP. The neural network maps the time t to the state at this time: $\mathbf{y}(t)$.

4.1.1. Loss Function

The loss function for PINNs consists of two terms basically:

$$MSE = MSE_y + MSE_F \tag{3}$$

While the first term, MSE_y, serves to impose the initial condition on the neural network by utilizing input data with corresponding output targets, the second term, MSE_F, is used to impose the physics of the system on the neural network output by penalizing it for not satisfying the ODE. The latter term acts as a regularization term by imposing a specific solution on the neural network. In contrast, the goal of a traditional regularization term is usually to penalize large magnitudes of the weights.

Initial Conditions

To impose the initial condition on the neural network, we create a training data point $(t, \hat{\mathbf{y}})$ consisting of an input $t = 0$ to the neural network and a corresponding target $\hat{\mathbf{y}}_0 = \mathbf{y}_0$ for its output. Then, MSE_y is calculated as follows:

$$MSE_y = \frac{1}{N_y} \left\| \mathbf{y}(0) - \hat{\mathbf{y}}_0 \right\|^2 \tag{4}$$

where: $\mathbf{y}(0)$ is the predicted output of the neural network for input $t = 0$; N_y is the dimension of $\mathbf{y}$ or the number of system states; and $\|\cdot\|$ is the ℓ_2-norm.

Physics

The neural network output should satisfy the ODE for the entire time horizon of interest $t \in [0, T]$ for the network to adhere to the physics of the underlying system. For this, N_f input points, $\{t^k : k = 1, \ldots, N_f\}$, are randomly sampled covering the entire time horizon $[0, T]$, yielding a dataset of collocation points. For each collocation point, a forward pass of the PINN is performed to calculate the corresponding output $\mathbf{y}(t^k)$. The resulting output

is used to calculate the deviation from the ODE in a residual function named **F**:

$$\mathbf{F}(\mathbf{y}) := \frac{\partial \mathbf{y}}{\partial t} - \mathbf{f}(\mathbf{y}) \tag{5}$$

where the first term is the neural network output $\mathbf{y}(t)$ differentiated with respect to its time input. This derivative is obtained using automatic differentiation, for instance, using TensorFlow. The second term of this equation, $\mathbf{f}(\mathbf{y})$, corresponds to the ODE calculated for $\mathbf{y}$. If these two terms match, our neural network's solution $\mathbf{y}$ satisfies the system model given by the ODE. This residual function $\mathbf{F}$ is applied to every collocation point and used to calculate the physics-related loss MSE_F:

$$MSE_F = \frac{1}{N_f} \sum_{k=1}^{N_f} \frac{1}{N_y} \left\| \mathbf{F}(\mathbf{y}(t^k)) \right\|^2 \tag{6}$$

The loss function defined in (3) can now be computed and used for training the PINN. If it can reproduce the initial condition, and the derivative of the neural network output satisfies the ODE for the entire time interval, $[0, T]$, then the output of the PINN is a solution to the IVP.

4.1.2. Training

To train PINNs, we must minimize the total loss function in (3). This minimization commonly employs a two-step strategy involving the Adam optimizer followed by the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimizer (Fletcher, 2000). The BFGS optimizer utilizes the Hessian matrix to determine the optimization direction, yielding more precise results by considering curvature in a high-dimensional space. However, if applied directly without preceding it with the Adam optimizer, there is a risk of rapid convergence to a local minimum for the residual without exploring other potential solutions. To mitigate this, we use the Adam optimizer initially to navigate away from local minima and subsequently refine the solution using the more accurate BFGS. In the context of Physics-Informed Neural Networks (PINNs), which often involve high-dimensional problems, the limited-memory version of BFGS (L-BFGS) is frequently employed to handle optimization problems with a large number of variables, as encountered in deep learning applications. This combined approach, involving a sequence of optimizers, has been successfully applied in physics-informed neural networks, particularly for solving systems of partial differential equations (PDEs), as

demonstrated in prior studies (Karniadakis et al., 2021). We apply the same methodology in the current study. Additionally, Taylor et al. (2022) present a recent investigation on different optimizers for PINNs.

4.2. Physics-Informed Neural Nets for Control (PINC)

In the traditional PINN framework, the initial condition is fixed and does not support a varying control input. The framework initially proposed in Antonelo et al. (2024) adds the initial condition and control signal as inputs to the PINN, leading to the Physics Informed Neural Networks for Control (PINC). This argumentation increases the input dimension of the PINN, along with the time required for training, enabling the resulting neural network to make predictions from any initial condition and control input. Figure 3a illustrates the concept whereby the PINC maps the input time, the initial condition, and control input to a state prediction at the input time t.
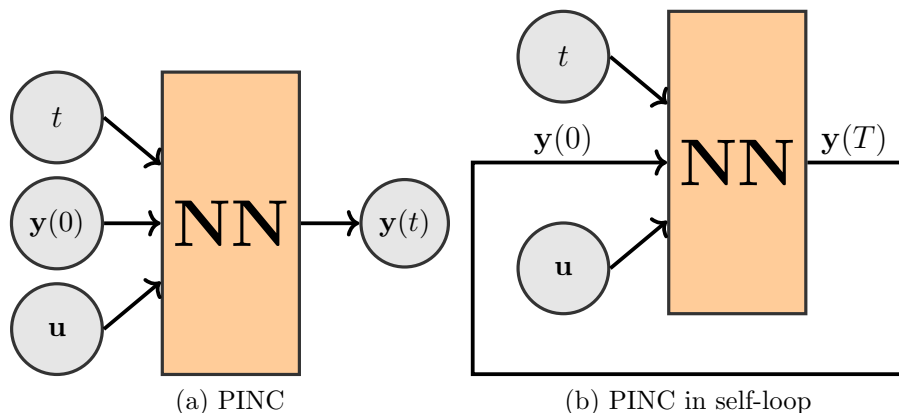

(a) PINC (b) PINC in self-loop

Figure 3: Physics-informed Neural Networks for Control (PINC). Left: PINC network with time t, initial condition $\mathbf{y}(0)$, and control input $\mathbf{u}$ as inputs. This neural network can make predictions of the state at time t given any initial condition and control input. Right: PINC network in self-loop mode, allowing for long-range simulations. The output state prediction at the time $t = T$ is fed back as the initial condition of the PINC to make a new prediction, progressing T seconds every iteration. A different control input may be applied at every iteration. For the first iteration, the initial condition must come from the outside, possibly measured from the actual system.

Some changes are needed for the training regime to account for the PINC's new inputs (see Figure 3). Namely, the training data input must be extended

with the additional inputs and their ranges selected. For the control input **u**, this should be the feasible range of the control input. The range for the initial condition should contain the states that the system can reach so that the neural network is trained to make predictions from any reachable state during operation.

4.2.1. Loss Function

The loss function for PINC is similar to the one for the traditional PINN, *i.e.*, one component for the initial conditions and one for the physics-based loss:

$$MSE = \lambda_y \cdot MSE_y + \lambda_F \cdot MSE_F \qquad (7)$$

The only difference here is that we added scaling factors to both terms, which may also be applied to the traditional PINN but was left out for easy understanding. Here, the scaling factors are scalars, but the concept may easily be extended to include individual scaling factors for each of the states so that both λ_y and λ_F can become vectors with the same dimension as **y**. This scaling allows the selection of priority for training of the different states, which may help obtain a better result.

Initial Conditions

For the traditional PINN, only one point was needed to train for the initial condition. Now, however, our initial condition belongs to a range of values, and we also need to consider different control inputs. We will create a set of training data points $\{\mathbf{v}^j : j = 1, \ldots, N_t\}$, where each point contains the inputs to the neural network $\mathbf{v}^j = \left[0, \mathbf{y}_0^j, \mathbf{u}^j\right]^T$. The first term is the time t, being 0 for all these data points because they represent the initial condition. $\mathbf{y}_0^j$ and $\mathbf{u}^j$ correspond to the j-th initial condition and control signal, randomly sampled within their feasible ranges to form N_t data points. The target for each of these training data points is the initial condition, which appears in the input: $\hat{\mathbf{y}}^j = \mathbf{y}_0^j$.

The loss for the initial conditions is then calculated as the MSE of all these training data points:

$$MSE_y = \frac{1}{N_t} \sum_{j=1}^{N_t} \frac{1}{N_y} \left\|\mathbf{y}(\mathbf{v}^j) - \hat{\mathbf{y}}^j\right\|^2 \qquad (8)$$

Physics

The collocation points, $\{\mathbf{v}^k : k = 1, \ldots, N_f\}$, used to impose the ODE on the PINC net also need to be extended with the new inputs, that is, $\mathbf{v}^k = \left[t^k, \mathbf{y}_0^k, \mathbf{u}^k\right]^T$. All three elements of each collocation point are sampled randomly from their respective domains.

The residual function needs a slight change to include the control input $\mathbf{u}$ in the ODE function:

$$\mathbf{F}(\mathbf{y}) := \frac{\partial \mathbf{y}}{\partial t} - \mathbf{f}(\mathbf{y}, \mathbf{u}) \tag{9}$$

The physics-related loss term MSE_F is the same as for the traditional PINN, only differing in the extended input to the neural network ($\mathbf{v}^k$):

$$MSE_F = \frac{1}{N_f} \sum_{k=1}^{N_f} \frac{1}{N_y} \left\| \mathbf{F}(\mathbf{y}(\mathbf{v}^k)) \right\|^2 \tag{10}$$

4.2.2. Long Range Simulations by Looping the PINC

The PINC is trained to make predictions within the time horizon $t \in [0, T]$, but longer-range simulations are possible by utilizing the PINC several times in a self-loop mode. First, we use the system's current state to make a prediction at time T; then, we use this prediction as input to the PINC to obtain a new prediction at time $2T$. We can continue this feeding back of the predicted state, simulating T seconds at every iteration until reaching the desired prediction time. Figure 3b illustrates how this works, feeding the neural network output back as the initial condition, input to the network, in the next time interval.

When used with MPC, the prediction time T of the PINC will be selected to match the length of a single step in the controller so that the PINC, with input $t = T$, will act as a function mapping the states between the time steps of the MPC. With this setup, the PINC allows choosing a different control input at every time step, which is perfect for MPC applications.

4.3. Model Predictive Control

In this work, Model Predictive Control (MPC) (Huba et al., 2011) will be employed to control an oil well. To ensure that the predictions of the MPC satisfy the system dynamics, we need a function that maps the current state and control input to the state at the next time step. For example, for a linear system, the prediction can be implemented by a state transition matrix

obtained from the discretization of the system. For a non-linear system, such as the well model, we can use a linearization of the model or a numerical integration method. Here, the PINC serves as the function mapping the current state and control input to the state at the next time step. As the PINC contains non-linear functions (the activation functions), the resulting MPC will be within the category called Non-linear MPC (NMPC).

In this work, the following formulation is considered for model predictive control of an oil well:

$$\min \sum_{i=1}^{N} (\mathbf{y}[k+i] - \mathbf{y}^{\texttt{ref}})^T \mathbf{Q} (\mathbf{y}[k+i] - \mathbf{y}^{\texttt{ref}}) + \sum_{i=0}^{N_u-1} \Delta \mathbf{u}[k+i]^T \mathbf{R} \Delta \mathbf{u}[k+i] \quad (11a)$$

subject to:

$$\mathbf{x}[k+j+1] = \mathbf{F}(\mathbf{x}[k+j], \mathbf{u}[k+j]), \; j = 0, \ldots, N-1 \qquad (11b)$$

$$\mathbf{y}[k+j] = \mathbf{F_y}(\mathbf{x}[k+j], \mathbf{u}[k+j-1]), \; j = 1, \ldots, N \qquad (11c)$$

$$\mathbf{u}[k+j] = \mathbf{u}[k+j-1] + \Delta \mathbf{u}[k+j], \; j = 0, \ldots, N_u-1 \qquad (11d)$$

$$\mathbf{u}[k+j] = \mathbf{u}[k+N_u-1], \; j = N_u, \ldots, N-1 \qquad (11e)$$

$$\mathbf{h}(\mathbf{x}[k+j], \mathbf{y}[k+j], \mathbf{u}[k+j-1]) \leq \mathbf{0}, \; j = 1, \ldots, N \qquad (11f)$$

where: the prediction horizon of MPC is N steps; the control input is allowed to change for the first N_u steps; equation (11a) defines the cost function of the optimization problem, whereby the deviations of the output variables from their references and the changes in control input are penalized; $\mathbf{Q}$ is the weight matrix for the output deviations from their references; $\mathbf{R}$ is the weight matrix that penalizes the changes in control inputs for the first N_u iterations; equation (11b) ensures that the solution satisfies the dynamics of the system, *i.e.*, $\mathbf{F}$ represents the PINC which maps, one time-step ahead, states to the next ones; $\mathbf{F_y}$ in equation (11c) is a function that maps the state and control inputs to some output variable $\mathbf{y}$, which is the variable we wish to control (the states themselves or algebraic variables, such as the bottom-hole pressure in the oil well).

Equation (11d) ensures that the first N_u control inputs change by their respective $\Delta \mathbf{u}$ factor. While the control input is only allowed to change for the first N_u iterations, afterward, it remains constant and equal to the control input at time step N_u according to equation (11e). Finally, equation (11f) defines the inequality constraints for the controller. They can limit the value of the control inputs $\mathbf{u}$ and also set bounds on states or other variables of interest.

4.4. Improved PINC

The improvements proposed for the PINC architecture in this paper, which are presented below, consist of adding skip connections to improve training and avoid vanishing gradients and changing the model's ODE equation to make the training of PINCs with a physics-based loss function feasible.

4.4.1. Skip Connections

Since gradients can vanish unwittingly during PINC network training, which decreases prediction performance, especially for the particular application of oil well modeling, we propose to employ an improved PINC architecture in this section. Any strategy that improves the magnitude of the gradients can greatly help train the neural network. One approach is to use skip connections in the network architecture to avoid vanishing gradients, as proposed by Wang et al. (2021). Besides the fully connected dense network, their proposed structure adds two more layers called encoders. The input to these encoder layers is the neural network input. In contrast, the output of these encoders is used to calculate the activation of each layer in the fully connected network, except for the final layer. This skip-connection structure ensures that the output of each layer of the neural network is closely connected to the input layer. Equation (12) expresses the mathematical formulation of the proposed structure,

$$
\begin{cases}
\mathbf{U} = \phi\left(\mathbf{W}^1\mathbf{X} + \mathbf{b}^1\right) \\
\mathbf{V} = \phi\left(\mathbf{W}^2\mathbf{X} + \mathbf{b}^2\right)
\end{cases}
\tag{12a}
$$

$$
\begin{cases}
\mathbf{Z}^{(1)} = \phi\left(\mathbf{W}^{z,1}\mathbf{X} + \mathbf{b}^{z,1}\right) \\
\mathbf{A}^{(1)} = \left(1 - \mathbf{Z}^{(1)}\right) \odot \mathbf{U} + \mathbf{Z}^{(1)} \odot \mathbf{V}
\end{cases}
\tag{12b}
$$

$$
\begin{cases}
\mathbf{Z}^{(k)} = \phi\left(\mathbf{W}^{z,k}\mathbf{A}^{(k-1)} + \mathbf{b}^{z,k}\right), \\
\mathbf{A}^{(k)} = \left(1 - \mathbf{Z}^{(k)}\right) \odot \mathbf{U} + \mathbf{Z}^{(k)} \odot \mathbf{V}, \quad k = 2, \dots, N_L
\end{cases}
\tag{12c}
$$

$$
\mathbf{y} = \mathbf{W}\mathbf{A}^{(N_L)} + b
\tag{12d}
$$

where $\odot$ denotes the Hadamard product, namely the element-wise product of two matrices. The visual representation of the structure of skip connections is depicted in Figure 4.

This structure implements a neural network with N_L hidden layers of N_n neurons each. $\mathbf{X}$ is the input to the network, which, for our application, is a vector containing the time, the initial condition, and the control input. The input $\mathbf{X}$ is projected into a higher dimensional space through the encoders
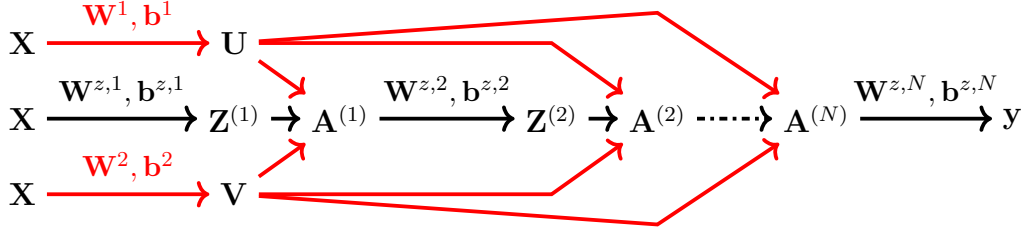
Figure 4: Illustration of the improved neural network architecture for PINNs, as proposed by Wang et al. (2021). **X** is the input to the neural network, which is projected into a higher dimensional space through the two encoder layers, resulting in the higher dimensional vectors **U** and **V**. For each layer of the main network, we first calculate the intermediate activation **Z** utilizing the weight matrix and bias vector of this layer, then **Z** is weighted by the encoder outputs **U** and **V** to calculate the final activation **A** of the layer.

layers by using their respective weights and biases $\mathbf{W}^1$, $\mathbf{W}^2$, $\mathbf{b}^1$, and $\mathbf{b}^2$, resulting in the N_n dimensional vectors $\mathbf{U}$ and $\mathbf{V}$. ϕ is the hyperbolic tangent activation function.

The first step for calculating the forward pass is a standard pass through a dense layer to compute $\mathbf{Z}^{(1)}$, as shown in Equation (12b). Then this $\mathbf{Z}^{(1)}$ is weighted by element wise multiplication ($\odot$) with the encoder vectors $\mathbf{U}$ and $\mathbf{V}$, to calculate the activation $\mathbf{A}^{(1)}$ of the first layer. This propagation continues through all the remainder of the N_L layers. The final output $\mathbf{y}$ is calculated as a linear projection of the previous layer's activation without the weighting from the encoder layers.

4.4.2. Safeguarding the Learning Process

Some of the ODE equations used to calculate MSE_F can lead to problems during training, so some modifications are needed for the neural network to train reliably. Modifying the ODE equations constitutes a problem that depends on the type of system we are interested in modeling with the PINC network. It happens that the oil well's ODEs have undesirable functions that can cause the training process to crash. Two of these problems are addressed in the following.

Tubing Friction Factor: Equation (A.6b) is highly non-linear and not defined for negative numbers inside the logarithm function. For the logarithm argument to be negative, the Reynolds number in the tubing must be sufficiently negative, which is not physically possible. However, such an undesirable condition arises during the initial phase of the training. To sim-

plify this equation and help accelerate learning, we replace the equation with an approximation using a third-order polynomial in the region of interest of Reynolds numbers, which can be found through test simulations.

During the initial training phase, the neural network will output poor predictions that can be far from physically feasible values. We can then end up with Reynolds numbers outside the interval for which the approximation was fit. The most straightforward tweak is to truncate the Reynolds number (calculated from the PINC prediction) to the selected interval. Because this truncation eliminates the dependency on the preceding variables, it also reduces the quality of the gradients when the Reynolds number gets truncated, but, in practice, the truncation works just fine. Equation (13a) formalizes the approximation resulting from the truncation of the Reynolds number, with $g(\cdot)$ being the third-order polynomial approximation defined in Equation (13b).

$$\lambda_{\mathtt{tb}}(Re_{\mathtt{tb}}) = \begin{cases} g(Re_{\mathtt{tb,min}}), & Re_{\mathtt{tb}} < Re_{\mathtt{tb,min}} \\ g(Re_{\mathtt{tb}}), & Re_{\mathtt{tb,min}} \leq Re_{\mathtt{tb}} \leq Re_{\mathtt{tb,max}} \\ g(Re_{\mathtt{tb,max}}), & Re_{\mathtt{tb}} > Re_{\mathtt{tb,max}} \end{cases} \tag{13a}$$

$$g(x) = ax^3 + bx^2 + cx + d \tag{13b}$$

The numerical values for the polynomial coefficients and the ranges of Reynolds numbers used to generate these approximations are shown in Table B.6. Appendix B provides more details on the implemented approximation. The oil well model also contains a friction factor for the bottom-hole friction, utilizing the same equations. However, since the flow from the reservoir is constant when calculating the Reynolds number here, this friction factor is also constant and no changes are needed.

Square Root: The square root is another function that can cause numerical problems, mainly during the initial phase of the training, because it is not defined for negative numbers. If the neural network outputs state predictions that result in a negative value in one of the square roots in the ODE, the resulting value will not be defined ("NaN" in Python). This "NaN" (Not a Number) will propagate through the calculations, invalidating the whole training.

The issue is resolved by simply applying the $\max(\cdot)$ operator within these functions to avoid negative values. Even though this tweak results in no gradients for negative numbers, it worked well since this mainly occurs in the initial iterations of the training. To avoid zero division error in other

17

equations, it is helpful to set the lower limit of the square root input to some small positive value (*e.g.*, 10^{-3}), a threshold that can be found by trial and error. This modified square root function is:

$$f_{\mathtt{sqrt}}(x) = \sqrt{\max\{x, 10^{-3}\}} \qquad (14)$$

4.5. Neural Network for Algebraic Variables

As we are interested in controlling the algebraic states of the oil well model, for instance, the bottom-hole pressure or liquid production, we need some means for obtaining the values of these variables as the current PINC framework only outputs the state predictions.

Although PINC could be trained to output the algebraic variables directly, this would limit its flexibility, as additional algebraic variables would require fully retraining the network from scratch. Thus, we propose to train an independent, traditional feedforward neural network to predict the algebraic variables **z** from the states **y**. The input to this neural network are the states **y** and the control input **u**, while the output is one or more of the algebraic variables **z** of the model. This setup is illustrated in Figure 5. This new neural network is trained independently of the PINC. The hierarchical architecture allows utilizing different algebraic variables in the MPC by simply training a new neural network without retraining the PINC model, which is much slower and more complex.
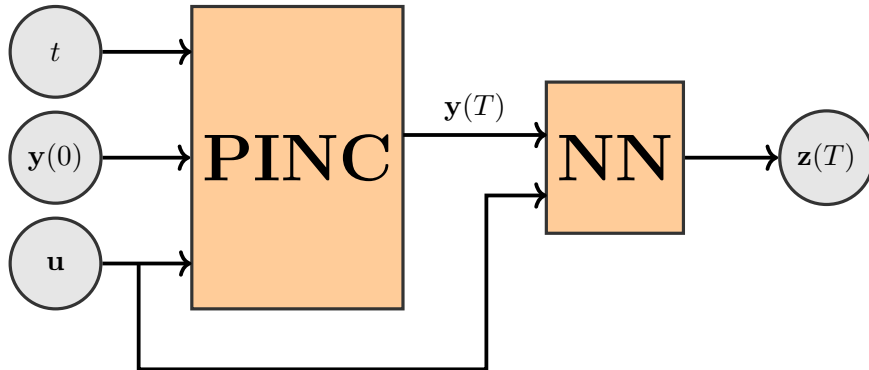

Figure 5: Hierarchical architecture for computing algebraic variables from the states. An additional, independently trained neural network is trained to predict the algebraic variables of the oil well, for instance, the bottom-hole pressure. The inputs to this supporting network are the state predictions from the PINC and the control input.

As no dynamics are involved in calculating the algebraic variables from the states, the PINN framework is unnecessary. Like the PINC, training data

is sampled from the entire state and control-input space. The corresponding target outputs (the values of the algebraic variables) are calculated using the ODE equations. It is important to note that modeling the entire system in an end-to-end manner with neural networks, *i.e.*, mapping inputs to the desired algebraic variables, allows the MPC implemented in CasADi[1] to leverage a smooth and differentiable function (*i.e.*, the neural network itself) as the system's model. Thus, using a network model enables the computation of a Jacobian without hindrances. CasADi solves the resulting NLP problem with IPOPT (Wächter and Biegler, 2006), an interior-point algorithm developed for problems with smooth and twice differentiable objective functions and constraints. IPOPT is a robust NLP solver widely applied in NMPC and optimal control.

Empirical observations revealed that relying solely on the ODE equations to compute algebraic variables from the PINC's state prediction presents challenges for the MPC in CasADi. In practice, this approach proved ineffective, as the optimization algorithm in CasADi often crashed before completing a meaningful simulation, with no discernible pattern to the crashes. It is likely that the nature of the equations, involving square roots, logarithms, and maximum operators, occasionally caused the optimizer to crash. This observation further underscores the rationale behind adopting the proposed hierarchical architecture.

Particularly for our oil well application, this supporting neural network will be trained to predict four algebraic variables based on the input from PINC's state predictions and control input, *i.e.*, the desired output $\mathbf{z}$ is a four-dimensional vector:

- *bottom-hole pressure* ($P_{\mathtt{bh}}$): the subject of control for one MPC application.

- *Gas-lift injection rate* ($w_{\mathtt{G,in}}$): a limited resource the controller needs to take into account.

- *Gas production* ($w_{\mathtt{G,res}}$): it may need to be constrained as some plants have limited capacity for processing gas.

- *Liquid production* ($w_{\mathtt{L,res}}$): it can be used for economic optimization.

---

[1]CasADi is an open-source tool for nonlinear optimization and algorithmic differentiation, which can be downloaded from https://web.casadi.org/

4.6. PINC for the Oil Well

Figure 5 depicts the architecture chosen for the PINC, where the PINC module employs the structure with skip connections. This PINC module has three types of inputs: time t; two control inputs corresponding to the production choke opening u_1 in (A.1f) and the gas-lift valve opening u_2 in (A.1a); and an initial condition corresponding to a three-dimensional vector with the mass of gas in the annulus $m_{\mathtt{G,an}}$, mass of gas in the tubing $m_{\mathtt{G,tb}}$, and mass of liquid in the tubing $m_{\mathtt{L,tb}}$. The physics-informed loss function corresponds to Eq. (1) and equations from Appendix A. In total, the input layer has six units, while the output layer has three units mapping the states $m_{\mathtt{G,an}}$, $m_{\mathtt{G,tb}}$, and $m_{\mathtt{L,tb}}$ at time t.

The NN module has a three-dimensional input layer, which receives the output prediction of the PINC module of the same dimension, *i.e.*, the predicted states $m_{\mathtt{G,an}}$, $m_{\mathtt{G,tb}}$, and $m_{\mathtt{L,tb}}$ at time t. It outputs a four-dimensional vector consisting of the algebraic variables described in the previous section, namely $P_{\mathtt{bh}}$, $w_{\mathtt{G,in}}$, $w_{\mathtt{G,res}}$, and $w_{\mathtt{L,res}}$. The bottom-hole pressure $P_{\mathtt{bh}}$ will be subject to control in our experiments by manipulating the two control inputs (valve openings).

4.7. Evaluation Metrics

4.7.1. Model Validation

During hyperparameter search, the trained PINC is evaluated on a validation set to select the hyperparameter values that yield the best network performance on unseen data. Thus, the MSE between the PINC output and the Runge-Kutta (RK) simulation of the gas-lifted oil well, which represents the actual system, defines the validation loss, namely:

$$MSE_{\mathtt{val}} = \frac{1}{N_{\mathtt{val}}N_y} \sum_{i=1}^{N_{\mathtt{val}}} \left\| \mathbf{N}(T, \mathbf{y}_{0,\mathtt{val}}^i, \mathbf{u}_{\mathtt{val}}^i) - \hat{\mathbf{y}}_{\mathtt{val}}^i \right\|^2 \tag{15}$$

where: N_y is the number of states of the model; $N_{\mathtt{val}} = 100$ is the number of validation points drawn randomly considering the chosen ranges for the initial conditions $\mathbf{y}_{0,\mathtt{val}}^i$ and control inputs $\mathbf{u}_{\mathtt{val}}^i$. The target points $\hat{\mathbf{y}}_{\mathtt{val}}^i$ are feasible points obtained from RK simulation; $\mathbf{N}(T, \mathbf{y}_{0,\mathtt{val}}^i, \mathbf{u}_{\mathtt{val}}^i)$ is the PINC network prediction $T = 60$ seconds ahead in time for validation point i; and $\|\cdot\|$ is the ℓ_2-norm.

Due to the issue of running into negative values in square roots, the self-looping simulation was not performed for model evaluation as proposed in

(Antonelo et al., 2024), since these simulations would quickly run into the infeasible state for our application. We assume that if PINC can yield good predictions from any initial condition (within the selected domain), it should also perform well in self-loop.

4.7.2. Evaluation of Prediction and Control

The evaluation of PINC predictions consists of iterating the network in self-loop mode, starting from random initial conditions and with a random sequence of control inputs, and then comparing its prediction to a Runge-Kutta simulation of the system. As we consider the ODE to represent the system perfectly, the Runge-Kutta simulation will represent the actual behavior of the oil well. The performance will be evaluated both visually and using the Integral of Absolute Error (IAE) on C sampling points along the simulation:

$$IAE = \frac{1}{C} \sum_{k=1}^{C} \frac{1}{N_y} \left\| \mathbf{y}[k] - \mathbf{r}[k] \right\|, \tag{16}$$

where $\mathbf{y}[k]$ is the output of the PINC network at the sample point k and $\mathbf{r}[k]$ is the Runge-Kutta reference simulation at the same sample point.

5. Experimental Setup

An improved PINC with skip connections, as detailed in the previous section, will be selected as a model of a gas-lifted oil well. Then, the control of the bottom-hole pressure of the oil well is tackled by embedding the trained PINC in a nonlinear MPC task.

This section presents the techniques and configurations implemented for training a PINC network with the best hyperparameters and the setup for applying nonlinear MPC using the trained PINC network.

5.1. Experimental Setup of PINC

5.1.1. Domain of Training Data

The training data and collocation points are generated as presented in Section 4.1.1. In addition, the upper limit for the time input t should be the prediction time of interest. For our application, the PINC prediction time is the time of a single step of the MPC, *i.e.*, $T = 60$ seconds. For the control input, the range corresponds to the feasible valve openings (production choke and gas-lift valve openings). The range for the states was selected by running

several Runge-Kutta simulations with different control inputs to determine the reachable states during operation. For the oil well model, the resulting domain of these data is a cube in the three-dimensional state space ($m_{\texttt{G,an}}$, $m_{\texttt{G,tb}}$, and $m_{\texttt{L,tb}}$), which may be suboptimal, as some volumes within this cube may be infeasible (*e.g.*, square root of a negative number). Infeasible points in the randomly drawn points from this cube are removed by running them through the ODE to check their feasibility. Excluding these infeasible points from the cube yielded the final training set for the states. Removing these infeasible points was very important, especially for the second well model.

For the initial condition (state), the procedure above is directly applied to generate the respective training set. For the collocation points, however, this procedure only verifies that the initial condition is feasible, not that the desired output prediction 60 seconds ahead of time exists. So, if the initial condition is close to infeasibility and the control input is unfortunate, the desired output may not be feasible. In practice, this works fine, but improvements may be possible.

5.1.2. Hyperparameters Search

During the hyperparameter search, all tentative configurations were trained 10 times with distinct initializations of the network weights and biases. For performance evaluation, the error on the validation set was considered.

Adam Learning Rate

For this search, we trained a PINC net with 5 hidden layers consisting of 20 neurons each for 1,000 iterations using the following learning rates: $\{1, 4, 7\} \times 10^{-4}$, $\{1, 4, 7\} \times 10^{-3}$, and $\{1, 4\} \times 10^{-2}$, resulting in the best choice of 7×10^{-3}, which presented the lowest validation prediction error.

Number of Training Data Points and Collocation Points

To find the number N_t of data points for the initial condition and the number N_f of collocation points, the same network of 5 hidden layers of 20 neurons each was employed. For each training run, after 1,000 epochs of Adam[2], other 1,000 iterations of L-BFGS followed. Figure 6 shows plots of

---

[2]On average, using the Adam method in the first thousand iterations yielded lower prediction validation error and faster training than using only L-BFGS optimization through the whole training, although both errors were in the same order of magnitude.
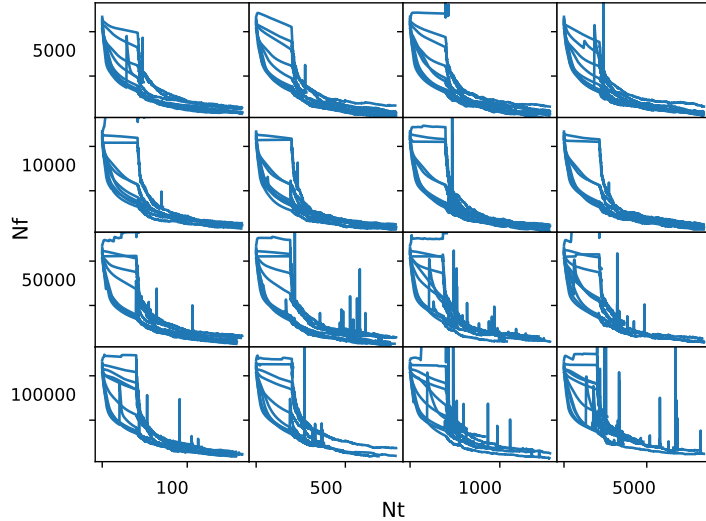
Figure 6: Validation error progression during PINC training with different combinations for the values of N_t (no. of data points) and N_f (no. of collocation points). In each subplot, the lines represent the training evolution of ten randomly initialized neural networks. The limits on the y-axis are the same for all the subplots. The best average performance was attained with N_t=1,000 and N_f=10,000, wherein none of the training attempts terminated prematurely.

the validation error during training of ten models for each combination of N_t and N_f. The y-axis has the same limits in all the plots. Some training experiments terminate prematurely due to converging to a poor local minimum. Considering both the final validation error and the number of training trials that fail, the best choice was the combination N_t=1,000 and N_f=10,000.

PINC Architecture Search

To find the right number of hidden layers and neurons per layer, a grid search was executed for $\{2, 4, 6, 8, 10\}$ layers and $\{10, 20, 30, 40\}$ neurons per layer, where each network was trained for 1,000 epochs with Adam followed by 1,000 iterations with L-BFGS (Figure 7). In a second step, the best three networks $(6 \cdot 30, 6 \cdot 40$ and $10 \cdot 30$ layers and neurons) were trained for additional 1,000 iterations with L-BFGS, yielding the architecture of 6 hidden layers of 30 neurons each with the lowest average validation loss.
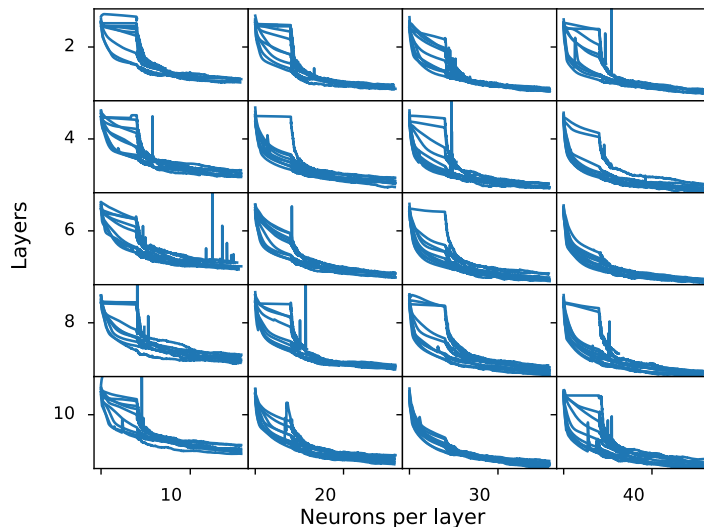
23

Figure 7: Search for the PINC architecture, presenting the validation error during the training of ten neural networks for each configuration, consisting of the number of hidden layers and neurons per layer. The best three networks are $6 \cdot 30$, $6 \cdot 40$, and $10 \cdot 30$ layers and neurons per layer. The y-axis has the same limits in all the subplots.

5.2. Experimental Setup of the NN for Algebraic Variables

We performed a hyperparameter search to find the best NN architecture for predicting the algebraic variables (the supporting neural network from Figure 5). The structures tested were fully densely connected neural networks with 1-4 hidden layers, each containing 10, 20, 30, and 40 neurons. Ten tests were conducted for each configuration, with different weight initializations, and trained for 5,000 iterations of L-BFGS. The validation set was constructed by randomly sampling 10,000 points from the input space, where the respective target values resulted from solving the oil-well ODE equations. The network of 4 hidden layers of 30 neurons each achieved the lowest validation error and was subsequently trained until convergence with L-BFGS. However, to speed up the MPC, a smaller network could be employed with no significant decrease in performance.

5.3. MPC Setup

The MPC controller was implemented in CasADi, an open-source numerical optimization framework[3].

The prediction horizon N of MPC depends on the control problem, which was chosen so that the system reaches a steady state within the specified prediction horizon. The time step of the predictions is $T = 60$ seconds for all control applications.

In typical applications, the optimization system defines a steady state, maximizing total oil production while adhering to system and physical constraints (Müller et al., 2022). The optimization system informs the resulting bottom-hole pressure to the control system as a reference for tracking. This pressure reaches its reference quickly and usually before the states stabilize. This fast reference tracking happens because of infinitely many combinations of state values and control inputs that will keep the bottom-hole pressure at the reference. Thus, a prediction horizon of $N = 50$ minutes is needed in this case. If the states run too far off, the system may end up in a region of the state space where the model is not defined, or the MPC cannot find a solution to the optimization problem. To ensure that a steady state is reached at the end of the prediction horizon, the control-input horizon is selected shorter ($N_u = 45$) than the prediction horizon ($N_u < N$). In practice, this means that the control input will be constant for the last 5 steps of the prediction horizon. Also, through the weight matrix $\mathbf{Q}$, an extra penalty is added to these final reference deviations (last 5 steps), making the MPC prioritize finding a stable state rather than damping the transient response:

$$Q_{ii} = \begin{cases} 1, & i \in \{1, \ldots, 45\} \\ 100, & i \in \{46, \ldots, 50\} \end{cases}, \quad R = \begin{bmatrix} 10^3 & 0 \\ 0 & 10^3 \end{bmatrix} \tag{17}$$

Appendix C gives more details on the MPC implementation in CasADi.

6. Results

Three variations of the oil well model, with different values of parameters as detailed in Appendix A, will be considered, but more results will be shown only with the first model for brevity. The experiments below refer to the first well model if not explicitly stated otherwise. Both the long-range prediction
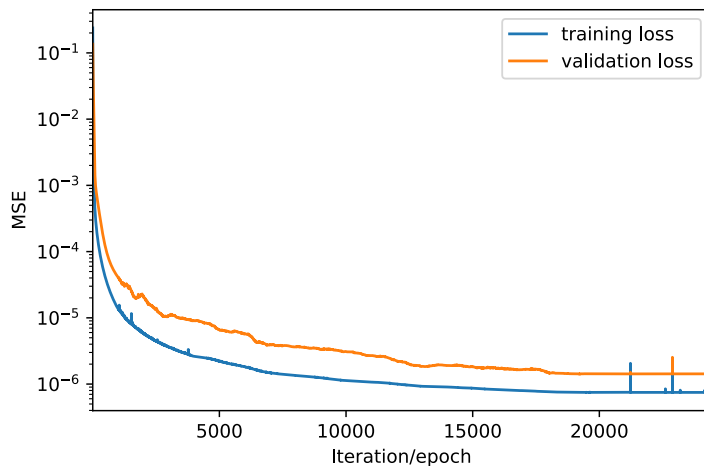

Figure 8: Plot of the training loss and validation error of the PINC trained for prediction of well 1. The PINC was trained for 1,000 epochs with Adam and then until convergence with L-BFGS. Because the nature of computation for the training loss (composed of the initial condition and physics-based loss) and validation error are distinct, the numeric values of these two losses are not directly comparable. The final training loss is $7.48 \cdot 10^{-7}$ and the validation loss, $1.43 \cdot 10^{-6}$.

and nonlinear MPC with the improved PINC for a gas-lifted oil well will be presented in this section.

6.1. Long-range Prediction for an Oil Well with the Improved PINC

The PINC used for prediction and control has 6 hidden layers of 30 neurons each, as found previously in the hyperparameter search. For the first well, Figure 8 reports the training loss and validation error during the training of its PINC. The validation loss shown in this figure consists of the average (scaled) error over 100 predictions 60 seconds ahead of time.

The performance of the trained PINC was evaluated for 3,000 seconds, with a sequence of random control inputs that changed every 60 seconds, using the IAE as a performance measure. The PINC was put in a self-loop for 50 iterations to obtain this simulation. Only the initial condition and the sequence of control inputs were input to the PINC network so that prediction errors would accumulate over the self-looping. Figure 9 shows the long-range PINC prediction for the first well, where the blue line corresponds to the Runge-Kutta simulation (represents the actual system's behavior), the dashed pink line gives the PINC prediction, and the larger pink dots
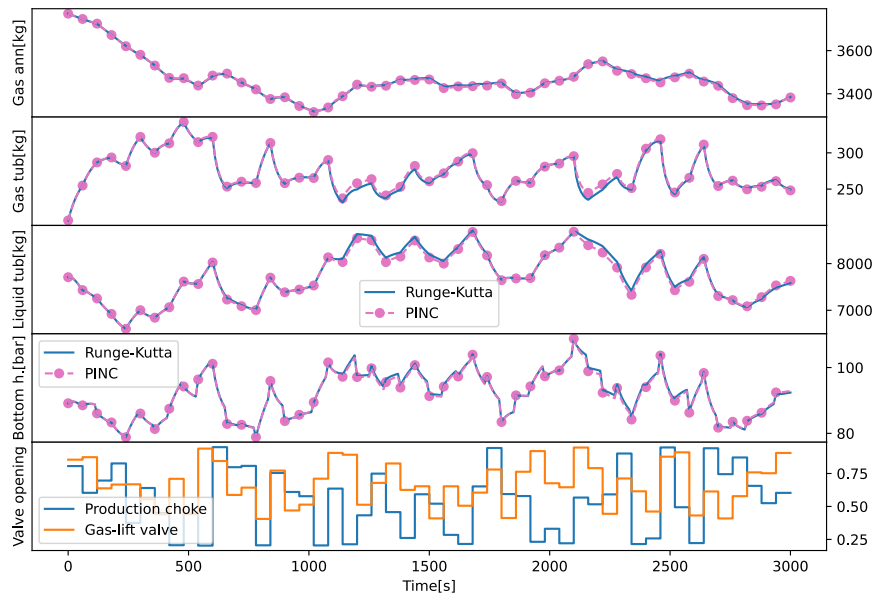
26

Figure 9: A test simulation of long-range PINC prediction for 3,000 seconds for the first well. A random sequence of control input, changing every 60 seconds, and a fixed (randomly chosen) initial condition are fed to the PINC. The resulting predictions are shown in pink, with the dots indicating each self-loop iteration. The blue line (in the upper three plots) shows the actual state values obtained by the Runge-Kutta simulation. The fourth plot shows the bottom-hole pressure predictions, generated by passing the PINCs predictions through the supporting neural network trained to predict the algebraic states. The bottom plot depicts the randomly generated control input sequence. The IAE for the states is 0.00959, a bit below the average, as shown in Table 2, while the IAE for the pressure is 0.2813 bar.

indicate the PINC output after each iteration of self-loop. While the first three uppermost plots report the state predictions produced by the PINC, the fourth one shows the bottom-hole pressure predictions generated by feeding the PINCs predictions through the supporting neural network, which predicts the algebraic states. The bottom-most plot depicts the randomly generated control-input sequence.

Table 2 shows the average and standard deviation of the IAE (of scaled variables) for 100 test simulations of long-range prediction (3,000 seconds each), where each test simulation resembles the one from Figure 9, but with different randomly generated control inputs. For the first well, the average and highest IAE of all three states are 0.00994 and 0.0332 bar, respectively.

Table 2: Average and standard deviation of the IAE of the prediction for all three states, and average IAE of the bottom-hole pressure prediction, for three oil wells for 100 test simulations of 3,000 seconds each, with randomly generated inputs. Only the pressure uses the second network for algebraic variables.

IAE	Well 1	Well 2	Well 3
average (states)	0.00994	0.00448	0.00185
standard deviation (states)	0.00621	0.00345	0.000325
average (pressure)	0.3087	0.2497	0.0656

Additionally, the average IAE of the bottom-hole pressure for the first well is 0.3087 bar.

6.2. Nonlinear MPC of Oil Well with Improved PINC

Here, experiments on tracking the bottom-hole pressure with the improved PINC and nonlinear MPC will be shown and compared to a linear MPC of a model linearized at each timestep, namely SLMPC (Appendix D). Figure 10 shows a simulation with two-step changes in the bottom-hole pressure reference. The states are shown in the figure to demonstrate that they settle after some time, even though they are not controlled or constrained in any way ($m_{\mathtt{G,an}}$, $m_{\mathtt{G,tb}}$, and $m_{\mathtt{L,tb}}$). The bottom-hole pressure follows the given reference very well for the PINC MPC controller, with an IAE of 0.038 bar. For comparison, SLMPC, a method that linearizes the ODE equation at every iteration of the control loop, described in Appendix D, is also plotted in blue, with an IAE of 0.058 bar. Notice that the biggest challenge in this experiment corresponds to the initial timesteps, where the bottom-hole pressure deviates from the reference for both controllers.

The same simulation was repeated several times with the addition of white measurement noise with a standard deviation of 5% added to the state measurements. Figure 11 shows an example of these simulations. Several of these simulations were completed, and the IAE was computed for each. The resulting average and standard deviation statistics are shown in Table 3, along with the IAE of the noiseless experiment. Regarding the simulations with measurement noise, Figure 12 reports the average response and the standard deviation, calculated along the trajectory. This figure shows in the solid line the average response, and the two shaded areas show one and two standard deviation regions, which encompass approximately 68% and 95% of the simulations, respectively.
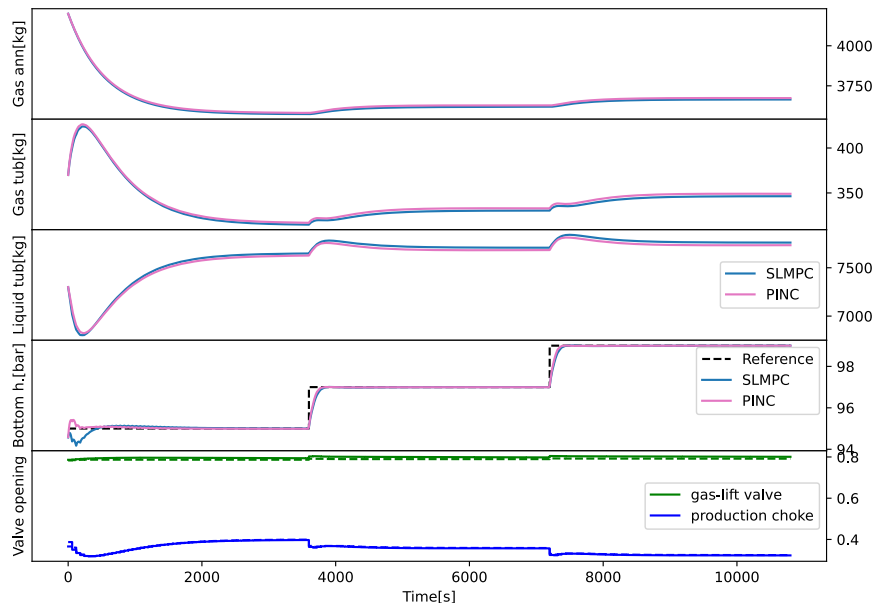
Figure 10: Controlling the bottom-hole pressure ($P_{\mathtt{bh}}$): simulation of three-step changes of 2 bar to the bottom-hole pressure reference (95 to 99 bar) for the PINC MPC in pink and the SLMPC in blue. For the control input in the bottom-most plot, the PINC MPC control input is shown in solid lines, and the SLMPC in dashed lines. The upper three plots show the states that are not controlled or constrained ($m_{\mathtt{G,an}}$, $m_{\mathtt{G,tb}}$, and $m_{\mathtt{L,tb}}$). The IAE of the PINC and SLMPC are 0.038 [bar] and 0.058 [bar], respectively.

6.3. Ablation of the Skip Connections

In this section, we investigate how skip connections affect the training of PINC. Figure 13 shows the Kernel Density Estimate (KDE) plot for the gradients of the residual $\mathbf{F}(\cdot)$ defined in (9), for two PINCs with six layers that are partially trained, one with skip connections (improved structure) and another without them (normal structure). The plots show the distribution of the gradients centered around zero for the weights and biases in all the layers. A high spike at the curve's center means that many of the gradients are near zero, which is undesirable. The orange line is flat whereas the blue line is vertical, showing that the gradients obtained with the improved structure are much better than the gradients of the normal structure, endorsing the use of skip connections to improve training. In particular, the improved PINC increased the magnitude of the gradient in the final layer by four orders of magnitude.
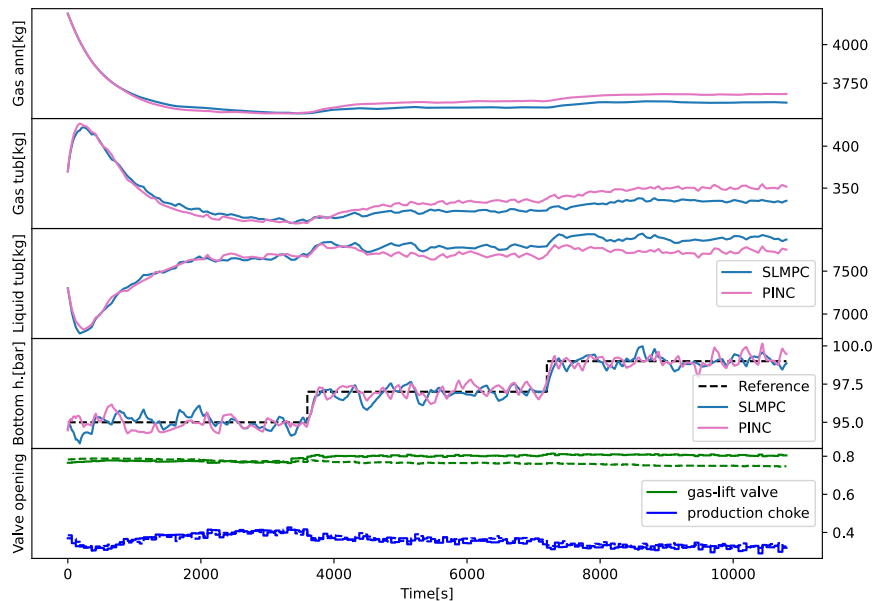
29

Figure 11: Controlling the bottom-hole pressure (P_{bh}) with the addition of measurement noise. In the bottom plot, the PINC MPC control input is shown in solid lines, and the linearized MPC is shown in dashed lines. The IAE of the PINC and SLMPC are 0.317 [bar] and 0.327 [bar], respectively.

Additionally, Figure 14 shows the validation error during training of 10 traditional networks in blue line and 10 networks with skip connections in gray, where the dots indicate the endpoint of the training. All networks were trained first with 500 epochs of Adam, followed by 500 iterations of L-BFGS. Six neural networks, three of each type, converged before completing the 500 iterations of L-BFGS. As the L-BFGS implementation evaluates the loss on average around three times per iteration, 500 iterations of L-BFGS translates to almost 2000 in the plot. The interesting part of Figure 14 is that for all seven networks where the training does not stop early, the new structure with skip connections outperformed the normal structure, reducing the validation error 67% on average. In our particular and challenging application of oil well modeling, the skip connections clearly improved training. Further, as the average success rate for training PINC without skip connections for the oil well is quite low, we have not shown the control results for this normal architecture.

Table 3: IAE of simulations with the two MPCs. The 0% noise experiment is the one shown in Figure 10, while the 5% noise statistics are calculated from several simulations, one of which is shown in Figure 11.

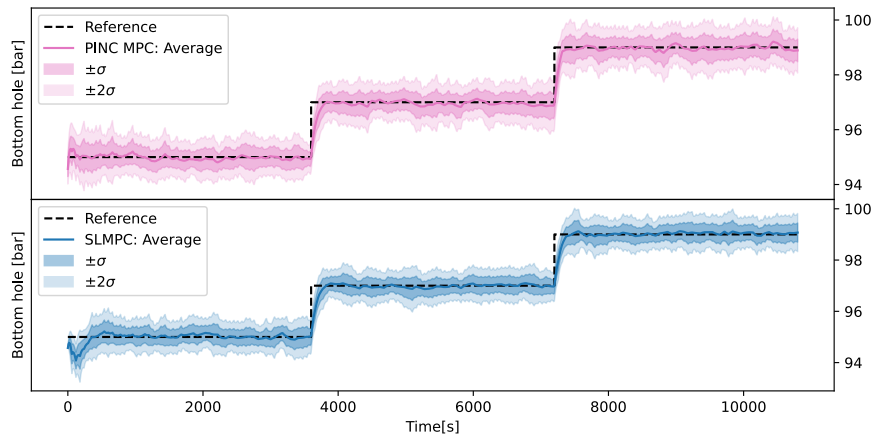| Noise		PINC MPC	SLMPC
0%	IAE	0.038	0.058
5%	IAE average	0.339	0.297
	IAE standard deviation	0.0200	0.0207

Figure 12: Controlling the bottom-hole pressure (P_{bh}) in several simulations with measurement noise: the average response and the standard deviation are plotted for both controllers. The upper (lower) plot shows the results of the PINC MPC (SLMPC) controller, where the innermost shaded area includes one standard deviation of the response (around 68% of the total), and the outer shaded area corresponds to two standard deviations of the response (around 95% of the total).

7. Conclusion

This work proposed an improved version of Physics-Informed Neural Networks for Control (PINC) of dynamic systems represented by ODEs. The improvements addressed a class of more realistic and complex dynamic systems, such as gas-lifted oil wells, characterized by highly nonlinear terms and functions not defined for negative numbers. These terms restrain the training as the gradients of the physics-loss function, which contains these terms, become less informative. The extensions proposed for PINC consist of (i) using a particular neural network architecture based on skip connections, which improves the magnitude of the gradients during the training of
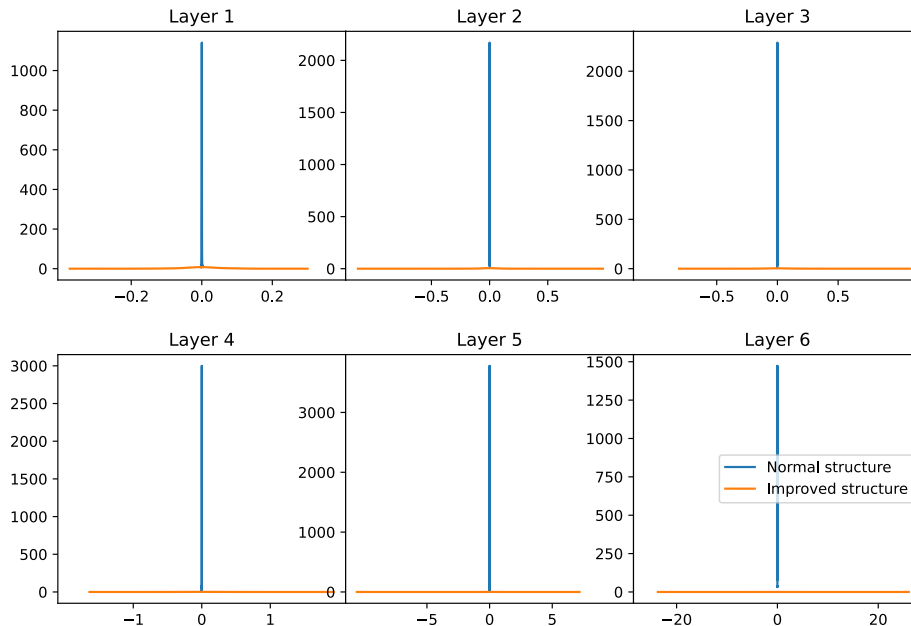
Figure 13: Kernel Density Estimate (KDE) plot comparing the gradients concerning the weights and biases of the residual function $\mathbf{F}(\cdot)$ defined in (9). The difference between the two PINC architectures is so expressive that the distributions of gradients for the normal structure (blue) seem like vertical lines compared to the orange horizontal lines for the improved structure (orange, with skip connections). For the last hidden layer (Layer 6), the average magnitude of the gradients for the normal structure is $2.16 \cdot 10^{-4}$, while for the improved structure, it is 2.55 (increasing by four orders of magnitude).

PINCs, avoiding known gradient pathologies in PINNs (Wang et al., 2021); (ii) modifying the ODE equations to avoid pitfalls for use in the loss function; and (iii) integrating a supporting feedforward neural network for learning the static mappings between system's states and algebraic variables.

Our proposed enhanced PINC achieved superior performance, averaging a 67% reduction in the validation prediction error for the oil well application compared to the original PINC. Additionally, it substantially improved the gradient flow through the network layers, increasing its magnitude by four orders of magnitude.

The results showed that the PINC framework can be successful in modeling and controlling complex systems with MPC, such as gas-lifted oil wells. PINC allowed for an open-ended long-range prediction of the system without access to the ODE (only used during training) and rendered the PINN
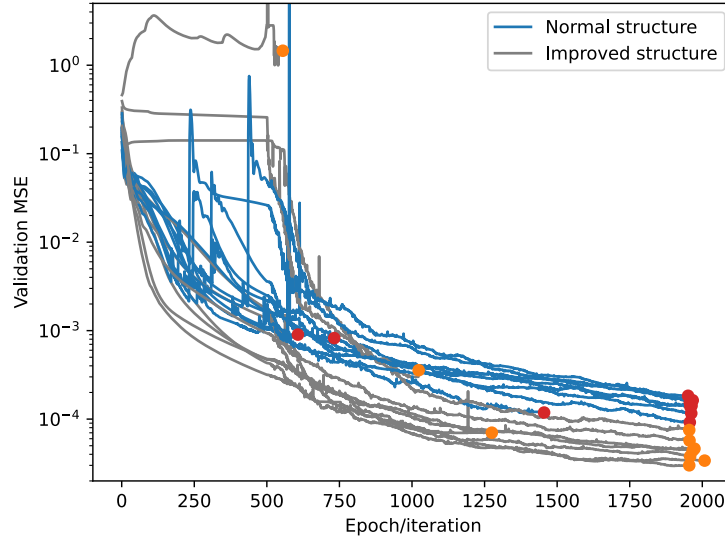
Figure 14: Results for the ablation of skip connections. Validation MSE through training of 20 neural networks, 10 with the normal dense structure and 10 with the improved structure (with skip connections). All networks were trained for 500 epochs of Adam, followed by 500 iterations of L-BFGS. The larger dots indicate the endpoints of the training, either by early convergence or by completing the 500 iterations of L-BFGS. Three networks of each type terminate training early. For the remainder, the improved neural network structure outperforms the standard/normal structure, reducing the validation error by 67% on average.

amenable to a control scenario where the bottom-hole reference was followed.

Future work will investigate using adaptive scaling factors λ_y and λ_F in the loss function, which is known to improve the training of PINNs (Xiang et al., 2022). Further, PINC could be applied to systems with other characteristics, for instance, where the reference changes continuously. The application of PINC to systems described by PDEs is also a research topic for future investigation.

Acknowledgments

Table A.4: Parameters for the oil well model from Jahanshahi et al. (2012).

Symb.	Description	Values	Units
R	universal gas constant	8.314	$\mathrm{J/(mol \cdot K)}$
g	gravity	9.81	$\mathrm{m/s^2}$
μ	viscosity	3.64×10^{-3}	$\mathrm{Pa \cdot s}$
$\rho_{\mathtt{L}}$	liquid density	760	$\mathrm{kg/m^3}$
$M_{\mathtt{G}}$	gas molecular weight	0.0167	$\mathrm{kg/mol}$
$T_{\mathtt{an}}$	annulus temperature	348	K
$V_{\mathtt{an}}$	annulus volume	64.34	$\mathrm{m^3}$
$L_{\mathtt{an}}$	annulus length	2048	$\mathrm{m^3}$
$P_{\mathtt{gs}}$	gas source pressure	140	bar
$S_{\mathtt{bh}}$	cross-section below injection point	0.0314	$\mathrm{m^2}$
$L_{\mathtt{bh}}$	length below injection point	75	m
$T_{\mathtt{tb}}$	injection point tubing temperature	369.4	K
GOR	mass gas oil ratio	0	–
$P_{\mathtt{res}}$	reservoir pressure	160	bar
$\overline{w}_{\mathtt{res}}$	average mass flow from reservoir	18	$\mathrm{kg/s}$
$D_{\mathtt{tb}}$	tubing diameter	0.134	m
$L_{\mathtt{tb}}$	tubing length	2048	m
$V_{\mathtt{tb}}$	tubing volume	25.03	$\mathrm{m^3}$
ϵ	piping superficial roughness	2.80×10^{-5}	m
PI	productivity index	2.47×10^{-6}	$\mathrm{kg/(s \cdot Pa)}$
$K_{\mathtt{gs}}$	gas-lift choke cons.	9.98×10^{-5}	–
$K_{\mathtt{inj}}$	injection valve cons.	1.40×10^{-4}	–
$K_{\mathtt{pr}}$	production choke cons.	2.90×10^{-3}	–

Appendix A. Well Model's Remaining Equations

Appendix A.1. Parameters of oil well models

Table A.4 shows the model parameters used for the primary oil well, which were extracted from Jahanshahi et al. (2012), except for ϵ which could not be found in the article. Instead, it was collected from Jordanou (2019). Several oil wells were constructed by altering the parameters of the model, whose values are shown in Table A.5.

Symb.	Well 1	Well 2	Well 3
R	8.314	8.314	8.314
g	9.81	9.81	9.81
μ	3.64×10^{-3}	3.64×10^{-3}	3.64×10^{-3}
ρ_L	760	760	730
M_G	0.0167	0.0167	0.0167
T_{an}	348	335	360
V_{an}	64.34	84.82	56.55
L_{an}	2048	2700	1800
P_{gs}	140	140	140
S_{bh}	0.0314	0.0314	0.0314
L_{bh}	75	75	40
T_{tb}	369.4	355.6	381.2
GOR	0	0	0.2
P_{res}	160	165	157
$\overline{w}_{res}$	18	11	30
D_{tb}	0.134	0.130	0.134
L_{tb}	2048	2700	1800
V_{tb}	25.03	31.00	22.08
ϵ	2.80×10^{-5}	2.80×10^{-5}	2.80×10^{-5}
PI	2.47×10^{-6}	2.12×10^{-6}	3.89×10^{-6}
K_{gs}	9.98×10^{-5}	10.43×10^{-5}	3.89×10^{-5}
K_{inj}	1.40×10^{-4}	1.20×10^{-4}	1.78×10^{-4}
K_{pr}	2.90×10^{-3}	2.43×10^{-3}	3.22×10^{-3}

Table A.5: Parameters for all the wells. First column is equivalent to the model from Table A.4

Appendix A.2. Mass flows

$$w_{\texttt{G,in}} = K_{\texttt{gs}} u_2 \sqrt{\rho_{\texttt{G,in}} \max(P_{\texttt{gs}} - P_{\texttt{at}}, 0)} \tag{A.1a}$$

$$w_{\texttt{G,inj}} = K_{\texttt{inj}} \sqrt{\rho_{\texttt{G,an,b}} \max(P_{\texttt{an,b}} - P_{\texttt{tb,b}}, 0)} \tag{A.1b}$$

$$w_{\texttt{res}} = PI \max(P_{\texttt{res}} - P_{\texttt{bh}}, 0) \tag{A.1c}$$

$$w_{\texttt{L,res}} = (1 - \alpha_{\texttt{G,tb,b}}^{\texttt{m}}) w_{\texttt{res}} \tag{A.1d}$$

$$w_{\texttt{G,res}} = \alpha_{\texttt{G,tb,b}}^{\texttt{m}} w_{\texttt{res}} \tag{A.1e}$$

$$w_{\mathtt{out}} = K_{\mathtt{pr}} u_1 \sqrt{\rho_{\mathtt{mix,tb,t}} \max(P_{\mathtt{tb,t}} - P_0, 0)} \tag{A.1f}$$

$$w_{\mathtt{L,out}} = (1 - \alpha_{\mathtt{G,tb,t}}^{\mathtt{m}}) w_{\mathtt{out}} \tag{A.1g}$$

$$w_{\mathtt{G,out}} = \alpha_{\mathtt{G,tb,t}}^{\mathtt{m}} w_{\mathtt{out}} \tag{A.1h}$$

The gas flow entering the annulus is $w_{\mathtt{G,in}}$ at the top, while $w_{\mathtt{G,inj}}$ is the flow of gas leaving the annulus and entering the production tubing. The total mass flow, liquid flow, and gas flow entering the production tubing from the reservoir are given by $w_{\mathtt{res}}$, $w_{\mathtt{L,res}}$, and $w_{\mathtt{G,res}}$ respectively. The total mass flow, liquid flow, and gas flow leaving the production tubing at the top-side are $w_{\mathtt{out}}$, $w_{\mathtt{L,out}}$, and $w_{\mathtt{G,out}}$ respectively.

Appendix A.3. Pressures

$$P_{\mathtt{an,t}} = \frac{RT_{\mathtt{a}} m_{\mathtt{G,an}}}{M_{\mathtt{G}} V_{\mathtt{a}}} \tag{A.2a}$$

$$P_{\mathtt{an,b}} = P_{\mathtt{an,t}} + \frac{m_{\mathtt{G,an}} g L_{\mathtt{an}}}{V_{\mathtt{an}}} \tag{A.2b}$$

$$P_{\mathtt{tb,t}} = \frac{\rho_{\mathtt{G,tb,t}} R T_{\mathtt{tb}}}{M_{\mathtt{G}}} \tag{A.2c}$$

$$P_{\mathtt{tb,b}} = P_{\mathtt{tb,t}} + \bar{\rho}_{\mathtt{mix}} g L_{\mathtt{tb}} + F_{\mathtt{tb}} \tag{A.2d}$$

$$P_{\mathtt{bh}} = P_{\mathtt{tb,b}} + F_{\mathtt{bh}} + \rho_{\mathtt{L}} g L_{\mathtt{bh}} \tag{A.2e}$$

The equations above define the pressure $P_{\mathtt{an,t}}$ at the top and the pressure $P_{\mathtt{an,b}}$ at the bottom of the annulus. Likewise, $P_{\mathtt{tb,t}}$ and $P_{\mathtt{tb,b}}$ correspond to the pressure at the top and bottom of the tubing, respectively. Finally, $P_{\mathtt{bh}}$ is the bottom-hole pressure.

Appendix A.4. Densities

The densities of gas and liquid in the streams in the annulus and tubing, when applicable, are given by the equations below.

$$\rho_{\mathtt{G,an,b}} = \frac{P_{\mathtt{an,b}} M_{\mathtt{G}}}{R T_{\mathtt{an}}} \tag{A.3a}$$

$$\rho_{\mathtt{G,in}} = \frac{P_{\mathtt{gs}} M_{\mathtt{G}}}{R T_{\mathtt{an}}} \tag{A.3b}$$

$$\rho_{\mathtt{G,tb,t}} = \frac{m_{\mathtt{G,tb}}}{V_{\mathtt{tb}} + S_{\mathtt{bh}}L_{\mathtt{bh}} - m_{\mathtt{L,tb}}/\rho_{\mathtt{L}}} \tag{A.3c}$$

$$\overline{\rho}_{\mathtt{mix}} = \frac{m_{\mathtt{G,tb}} + m_{\mathtt{L,tb}} - \rho_{\mathtt{L}}S_{\mathtt{bh}}L_{\mathtt{bh}}}{V_{\mathtt{tb}}} \tag{A.3d}$$

$$\rho_{\mathtt{G,tb,b}} = \frac{P_{\mathtt{tb,b}}M_G}{RT_{\mathtt{tb}}} \tag{A.3e}$$

$$\rho_{\mathtt{mix,tb,t}} = \alpha_{\mathtt{L,tb,t}}\rho_{\mathtt{L}} + (1 - \alpha_{\mathtt{L,tb,t}})\rho_{\mathtt{G,tb,t}} \tag{A.3f}$$

Appendix A.5. Mass/liquid fractions

$$\overline{\alpha}_{\mathtt{L,tb}} = \frac{m_{\mathtt{L,tb}} - \rho_{\mathtt{L}}S_{\mathtt{bh}}L_{\mathtt{bh}}}{V_{\mathtt{tb}}\rho_{\mathtt{L}}} \tag{A.4a}$$

$$\alpha_{\mathtt{G,bh}}^{\mathtt{m}} = GOR/(GOR+1) \tag{A.4b}$$

$$\alpha_{\mathtt{L,tb,b}} = \frac{w_{\mathtt{L,res}}\rho_{\mathtt{G,tb,b}}}{w_{\mathtt{L,res}}\rho_{\mathtt{G,tb,b}} + (w_{\mathtt{G,inj}} + w_{\mathtt{G,res}})\rho_{\mathtt{L}}} \tag{A.4c}$$

$$\alpha_{\mathtt{L,tb,t}} = 2\overline{\alpha}_{\mathtt{L,tb}} - \alpha_{\mathtt{L,tb,b}} \tag{A.4d}$$

$$\alpha_{\mathtt{G,tb,t}}^{\mathtt{m}} = \frac{(1 - \alpha_{\mathtt{L,tb,t}})\rho_{\mathtt{G,tb,t}}}{\alpha_{\mathtt{L,tb,t}}\rho_{\mathtt{L}} + (1 - \alpha_{\mathtt{L,tb,t}})\rho_{\mathtt{G,tb,t}}} \tag{A.4e}$$

Appendix A.6. Velocities

$$\overline{U}_{\mathtt{L,tb}} = \frac{4(1 - \alpha_{\mathtt{G,bh}}^{m})\overline{w}_{\mathtt{res}}}{\rho_{\mathtt{L}}\pi D_{\mathtt{tb}}^2} \tag{A.5a}$$

$$\overline{U}_{\mathtt{G,tb}} = \frac{4(w_{\mathtt{G,in}} + \alpha_{\mathtt{G,bh}}^{\mathtt{m}}\overline{w}_{\mathtt{res}})}{\rho_{\mathtt{G,tb,t}}\pi D_{\mathtt{tb}}^2} \tag{A.5b}$$

$$\overline{U}_{\mathtt{mix,tb}} = \overline{U}_{\mathtt{L,tb}} + \overline{U}_{\mathtt{G,tb}} \tag{A.5c}$$

$$\overline{U}_{\mathtt{L,bh}} = \frac{\overline{w}_{\mathtt{res}}}{\rho_L S_{\mathtt{bh}}} \tag{A.5d}$$

These equations express the liquid, gas and mix velocities in the tubing, and the liquid velocity in the bottom-hole.

Appendix A.7. Friction terms

$$Re_{\mathtt{tb}} = \frac{\overline{\rho}_{\mathtt{mix,tb}}\overline{U}_{\mathtt{mix,tb}}D_{\mathtt{tb}}}{\mu} \tag{A.6a}$$

$$\frac{1}{\sqrt{\lambda_{\mathtt{tb}}}} = -1.8\log_{10}\left[\left(\frac{\epsilon/D_{\mathtt{tb}}}{3.7}\right)^{1.11} + \frac{6.9}{Re_{\mathtt{tb}}}\right] \tag{A.6b}$$

$$F_{\mathtt{tb}} = \frac{\overline{\alpha}_{\mathtt{L,tb}}\lambda_{\mathtt{tb}}\overline{\rho}_{\mathtt{mix,tb}}\overline{U}^2_{\mathtt{mix,tb}}L_{\mathtt{tb}}}{2D_{\mathtt{tb}}} \tag{A.6c}$$

$$Re_{\mathtt{bh}} = \frac{\rho_{\mathtt{L}}\overline{U}_{\mathtt{L,bh}}D_{\mathtt{bh}}}{\mu} \tag{A.6d}$$

$$\frac{1}{\sqrt{\lambda_{\mathtt{bh}}}} = -1.8\log_{10}\left[\left(\frac{\epsilon/D_{bh}}{3.7}\right)^{1.11} + \frac{6.9}{Re_{\mathtt{bh}}}\right] \tag{A.6e}$$

$$F_{\mathtt{bh}} = \frac{\lambda_{\mathtt{bh}}\rho_{\mathtt{L}}\overline{U}^2_{\mathtt{L,bh}}L_{\mathtt{bh}}}{2D_{\mathtt{bh}}} \tag{A.6f}$$

These equations calculate the friction pressure-loss terms for the tubing (A.6c) and the bottom-hole (A.6f). For the bottom-hole friction calculations, the average mass flow from the reservoir, $\overline{w}_{res}$, is used, as can be seen in Equation (A.5d), resulting in Equations (A.6d), (A.6e) and (A.6f) being constant, which is another of the simplifications made in the derivation of the model to make it explicit.

Appendix A.8. Simplifications and Discussion

Simplifications were implemented in the algebraic equations in order to avoid implicit terms, resulting in an explicit model in the form of a system of ODE equations that can be simulated with methods such as Runge-Kutta.

Equations (A.5a), (A.5b) and (A.5d) use the predefined constant $\overline{w}_{\mathtt{res}}$ as the flow from the reservoir, which is one of the simplifications in the model derived in Jahanshahi et al. (2012) to avoid an implicit set of equations.

For the bottom-hole friction calculations, the average mass flow from the reservoir, $\overline{w}_{res}$, is used, as can be seen in Equation (A.5d), resulting in Equations (A.6d), (A.6e) and (A.6f) being constant, which is another simplification in the derivation of the model to render it explicit.

Equation (A.5b) relates the velocity of gas to the mass flow of gas injected into the tubing ($w_{\texttt{G,inj}}$), but the latter is replaced with the mass flow of gas injected into the annulus through the gas-lift choke ($w_{\texttt{G,in}}$). This allows changes in the gas-lift choke opening to directly affect the velocities in the tubing, the tubing friction loss, and the pressure at the bottom of the tubing, which happens more quickly than the change in pressure at the bottom of the annulus. The mass flow of gas injected into the tubing $w_{\texttt{G,inj}}$ is dependent on the pressure difference between the bottom of the annulus and the bottom of the tubing. A change in the gas-lift choke opening has an immediate effect on the gas injected into the tubing.

Table A.4 shows the model parameters of the primary oil well, which were extracted from Jahanshahi et al. (2012), except for ϵ which could not be found in the article and instead it was collected from Jordanou (2019).

Appendix B. Approximation of ODE of oil well model for PINC

Figure B.15 shows an example of the 3rd-order polynomial approximation of the friction factor $\lambda_{\texttt{tb}}$ as given by Equations (13a) and (13b), in comparison to the model Equation (A.4c). The range of Reynolds numbers for which the approximation is fit is: $[13000, 115000]$. This third-order approximation was found to resemble the original friction factor satisfactorily. It was verified by comparing Runge-Kutta simulations with the original equations and this approximation, in which there was no noticeable difference between the two.

The numerical values for the coefficients in Equation (13b) and the ranges of Reynolds numbers in Equation (13a) used to generate these approximations are shown in Table B.6.

Table B.6: Table presenting the numerical values for the third-order polynomial approximations for the three oil wells, along with the interval of Reynolds numbers for which the estimation was fitted. The bottom row shows the factor for which each column element should be multiplied, for example, for the first well: $a = -1.78 \cdot 10^{-17}$.

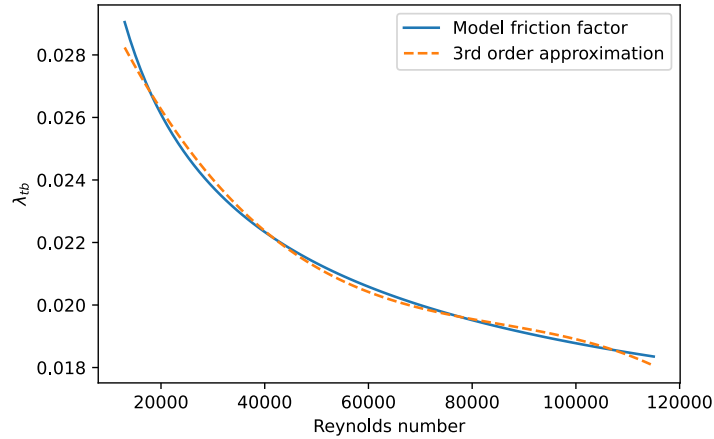|	a	b	c	d	$Re_{\texttt{tb,min}}$	$Re_{\texttt{tb,max}}$
Well 1	-1.78	4.56	-4.18	3.29	13000	115000
Well 2	-1.78	4.55	-4.17	3.29	13000	115000
Well 3	-0.203	0.887	-1.48	2.67	50000	160000
Factor	10^{-17}	10^{-12}	10^{-7}	10^{-2}		

Figure B.15: The third-order polynomial approximation of the tubing friction factor in Equation (A.6c) in orange compared with the real friction factor in blue. This plot shows the approximation for the first well, which was fitted on the interval $[13000, 115000]$ using the curve_fit function of SciPy.

Appendix C. MPC implementation in CasADi

The MPC will be formulated as a general Non-Linear Programming (NLP) problem using the casadi.Opti class. CasADi solves the resulting NLP problem with IPOPT (Wächter and Biegler, 2006). IPOPT is an interior-point algorithm developed for problems with smooth and twice differentiable objective functions and constraints, which is widely applied in NMPC and optimal control.

In the following code listing, line 1 creates an instance of this class, then variables and parameters are added to the Opti object:

```
opti = casadi.Opti()
N_state=3
N_input=2
x = opti.variable(N_state, N+1)
u = opti.variable(N_input, N)
du = opti.variable(N_input, N_u)    #change in u for each step
P_bh = opti.variable(1, N)

x0 = opti.parameter(N_state,1)
u_last = opti.parameter(N_input, 1)
P_bh_ref = opti.parameter(1, N)
Q = opti.parameter((N+1), (N+1))
```

```
13  R = opti.parameter(N_u*N_input, N_u*N_input)
```
Listing 1: Instantiating variables and parameters for MPC of the bottom-hole pressure.

Here, N_state is the number of states and N_input is the number of control inputs (production choke and gas-lift valve). This setup is for controlling the bottom-hole pressure. The opti.variable() type refers to variables of the optimization problem, while opti.parameter() regards the parameters given to the MPC at every iteration, such as reference and weights. x0 is the current measured (or estimated) state value, and u_last is the last applied control input, which is needed as we penalize the change in control input.

An important implementation detail regards ensuring that the variables x and P_bh satisfy the system dynamics. Code listing 2 shows that the function opti.subject_to() is used only on the state variable, which constrains the solution to satisfy the system dynamics. Here, F() represents a trained PINC network, a function that maps a state and control input to the state 60 seconds ahead of time. For the bottom-hole pressure, there is no need for a constraint; an assignment operator is enough. F_x_to_P_bh() represents the output of the second network trained to predict the algebraic variable bottom-hole pressure based on the values of the states and control input. By the opti.subject_to() here as well, we add unduly complexity to the optimization problem, which results in slower computation.

```
1  opti.subject_to(x[:,0] == x0)
2  for i in range(0,N):              # ODE constraint
3      opti.subject_to(x[:,i+1] == F(x[:,i],u[:,i])) # PINC
4
5  for i in range(0, N):
6      P_bh[:,i] = F_x_to_P_bh(x[:,i+1], u[:,i]) # NN algebraic
```
Listing 2: Ensuring that the MPC solution satisfy the system dynamics.

Constraints are also needed to enforce the relationship between the control inputs $u[k]$ and the change in control inputs $\Delta u[k]$, which is implemented by the following code:

```
1  opti.subject_to(u[:,0] == u_last + du[:,0])
2  for i in range(0, N_u-1):
3      opti.subject_to(u[:,i+1] == u[:,i] + du[:,i+1])
4  for i in range(N_u, N):
5      opti.subject_to(u[:,i] == u[:,N_u-1])
```
Listing 3: Implementing control input change constraints.

The first line ensures that the last applied control input is considered, while lines 2-3 enforce this relationship for the control input horizon. Finally, lines 4-5 fix the control input for the final steps if $N_u < N$. du is then used in the cost function to penalize the control input change.

For the first simulation iteration, we need to know the current control input or set the first elements of the **R** matrix to zero. This second option allows the first control input of the MPC prediction to take any value without being penalized. This flexibility can be useful when starting a simulation from an initial condition and when a steady-state control input is unknown.

Appendix D. Successive Linearization based MPC (SLMPC)

Successive Linearization based MPC (SLMPC) is a controller that, at every time step, linearizes the model around the operating point and solves a linear MPC problem. This kind of controller was successfully applied to chemical reactors (Seki et al., 2002) and variable stiffness actuated robots (Zhakatayev et al., 2017).

At every iteration of the MPC, the system equations are linearized and discretized around the current operating point $\mathbf{x}_k$, current control input $\mathbf{u}_k$, and current control variable $\mathbf{y}_k$. The linearized model can then be written as:

$$\Delta\mathbf{x}_{k+j+1} = \Delta\mathbf{x}_{k+j} + \mathbf{A}_k\Delta\mathbf{x}_{k+j} + \mathbf{B}_k\Delta\mathbf{u}_{k+j} + \delta_k \tag{D.1a}$$

$$\mathbf{y}_{k+j} = \mathbf{C}_k\Delta\mathbf{x}_{k+j} + \mathbf{D}_k\Delta\mathbf{u}_{k+j} + \mathbf{y}_k \tag{D.1b}$$

$$\Delta\mathbf{x}_{k+j} = \mathbf{x}_{k+j} - \mathbf{x}_k, \quad \Delta\mathbf{u}_{k+j} = \mathbf{u}_{k+j} - \mathbf{u}_k, \tag{D.1c}$$

where:

$$\mathbf{A}_k = T\frac{\partial\mathbf{f}}{\partial\mathbf{x}_k}(\mathbf{x}_k, \mathbf{u}_k), \ \mathbf{B}_k = T\frac{\partial\mathbf{f}}{\partial\mathbf{u}_k}(\mathbf{x}_k, \mathbf{u}_k), \tag{D.2a}$$

$$\delta_k = T\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \tag{D.2b}$$

$$\mathbf{C}_k = \frac{\partial\mathbf{h}}{\partial\mathbf{x}_k}(\mathbf{x}_k, \mathbf{u}_k), \ \mathbf{D}_k = \frac{\partial\mathbf{h}}{\partial\mathbf{u}_k}(\mathbf{x}_k, \mathbf{u}_k), \ \mathbf{y}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{u}_k), \tag{D.2c}$$

where T is the discretization time step length, $\mathbf{f}(\mathbf{x}, \mathbf{u})$ is the ODE system, and $\mathbf{h}$ is a function that computes the output from the states and control input: $\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{u})$. Then, the linear model in Equation (D.1a) can replace the PINC in the constraint of Equation (11b), and the linearized output Equation (D.1b) can be used to implement the constraint in Equation (11c).

References

Antonelo, E.A., Camponogara, E., Seman, L.O., de Souza, E.R., Jordanou, J.P., Hubner, J.F., 2021. Physics-informed neural nets for control of dynamical systems. URL: https://arxiv.org/abs/2104.02556, doi:10.48550/ARXIV.2104.02556.

Antonelo, E.A., Camponogara, E., Seman, L.O., de Souza, E.R., Jordanou, J.P., Hübner, J.F., 2024. Physics-informed neural nets for control of dynamical systems. Neurocomputing 579. doi:10.1016/j.neucom.2024.127419.

Camacho, E.F., Bordons, C., 2013. Model Predictive Control. Springer Science & Business Media.

Clarke, F.H., 1990. Optimization and Nonsmooth Analysis. SIAM. doi:10.1137/1.9781611971309.

Dwivedi, V., Srinivasan, B., 2020. Physics informed extreme learning machine (PIELM)–a rapid method for the numerical solution of partial differential equations. Neurocomputing 391, 96–118. doi:10.1016/j.neucom.2019.12.099.

Eren, U., Prach, A., Koçer, B.B., Raković, S.V., Kayacan, E., Açıkmeşe, B., 2017. Model predictive control in aerospace systems: Current state and opportunities. Journal of Guidance, Control, and Dynamics 40, 1541–1566. doi:10.2514/1.G002507.

Fletcher, R., 2000. Practical Methods of Optimization. John Wiley & Sons.

Fuks, O., Tchelepi, H.A., 2020. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. Journal of Machine Learning for Modeling and Computing 1, 19–37. doi:10.1615/JMachLearnModelComput.2020033905.

Gokhale, G., Claessens, B., Develder, C., 2022. Physics informed neural networks for control oriented thermal modeling of buildings. Applied Energy 314, 118852. doi:10.1016/j.apenergy.2022.118852.

Huba, M., Skogestad, S., Fikar, M., Hovd, M., Johansen, T., Rohal-Ilkiv, B., 2011. Selected Topics on Constrained and Nonlinear Control. STU/NTNU.

Jahanshahi, E., Skogestad, S., Hansen, H., 2012. Control structure design for stabilizing unstable gas-lift oil wells. IFAC Proceedings Volumes 45, 93–100. doi:10.3182/20120710-4-SG-2026.00110.

Jordanou, J.P., 2019. Echo State Networks for Online Learning Control and MPC of Unknown Dynamic Systems: Applications in the Control of Oil Wells. Master's thesis. Universidade Federal de Santa Catarina. URL: https://repositorio.ufsc.br/handle/123456789/214380.

Jordanou, J.P., Antonelo, E.A., Camponogara, E., 2022. Echo state networks for practical nonlinear model predictive control of unknown dynamic systems. IEEE Transactions on Neural Networks and Learning Systems 33, 2615–2629. doi:10.1109/TNNLS.2021.3136357.

Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L., 2021. Physics-informed machine learning. Nature Reviews Physics 3, 422–440. doi:10.1038/s42254-021-00314-5.

Liu, X.Y., Wang, J.X., 2021. Physics-informed Dyna-style model-based deep reinforcement learning for dynamic control. Proceedings of the Royal Society A 477, 20210618. doi:10.1098/rspa.2021.0618.

Müller, E.R., Camponogara, E., Seman, L.O., Hülse, E.O., Vieira, B.F., Miyatake, L.K., Teixeira, A.F., 2022. Short-term steady-state production optimization of offshore oil platforms: wells with dual completion (gas-lift and esp) and flow assurance. TOP 30, 152–180. doi:10.1007/s11750-021-00604-2.

Nascimento, T.P., Dórea, C.E.T., Gonçalves, L.M.G., 2018. Nonlinear model predictive control for trajectory tracking of nonholonomic mobile robots: A modified approach. International Journal of Advanced Robotic Systems 15. doi:10.1177/1729881418760461.

Nazari, L.F., Camponogara, E., Seman, L.O., 2023. Physics-informed neural networks for modeling water flows in a river channel. IEEE Transactions on Artificial Networks , 1–15. doi:10.1109/TAI.2022.3200028.

Nicodemus, J., Kneifl, J., Fehr, J., Unger, B., 2022. Physics-informed neural networks-based model predictive control for multi-link manipulators. IFAC-PapersOnLine 55, 331–336. doi:10.1016/j.ifacol.2022.09.117.

Ouyang, H., Zhu, Z., Chen, K., Tian, B., Huang, B., Hao, J., 2023. Reconstruction of hydrofoil cavitation flow based on the chain-style physics-informed neural network. Engineering Applications of Artificial Intelligence 119, 105724. doi:10.1016/j.engappai.2022.105724.

Raissi, M., 2018. Deep hidden physics models: Deep learning of nonlinear partial differential equations. Journal of Machine Learning Research 19, 1–24. URL: https://dl.acm.org/doi/pdf/10.5555/3291125.3291150.

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics 378, 686–707. doi:10.1016/j.jcp.2018.10.045.

Seki, H., Ooyama, S., Ogawa, M., 2002. Nonlinear model predictive control using successive linearization. Transactions of the Society of Instrument and Control Engineers 38, 61–66. doi:10.9746/sicetr1965.38.61.

Tang, Z., Dong, S., Yang, X., Zhang, J., 2023. Application of a parallel physics-informed neural network to solve the multi-body dynamic equations for full-scale train collisions. Applied Soft Computing 142, 110328. doi:10.1016/j.asoc.2023.110328.

Taylor, J., Wang, W., Bala, B., Bednarz, T., 2022. Optimizing the optimizer for data driven deep neural networks and physics informed neural networks. arXiv preprint arXiv:2205.07430 doi:10.48550/arXiv.2205.07430.

Wächter, A., Biegler, L.T., 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106, 25–57. doi:10.1007/s10107-004-0559-y.

Wang, S., Teng, Y., Perdikaris, P., 2021. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing 43, A3055–A3081. doi:10.1137/20M1318043.

Wang, S., Yu, X., Perdikaris, P., 2022. When and why PINNs fail to train: A neural tangent kernel perspective. Journal of Computational Physics 449, 110768. doi:10.1016/j.jcp.2021.110768.

Wu, P., Pan, K., Ji, L., Gong, S., Feng, W., Yuan, W., Pain, C., 2022. Navier–stokes generative adversarial network: a physics-informed deep learning

model for fluid flow generation. Neural Computing and Applications 34, 11539–11552. doi:`10.1007/s00521-022-07042-6`.

Xiang, Z., Peng, W., Liu, X., Yao, W., 2022. Self-adaptive loss balanced physics-informed neural networks. Neurocomputing 496, 11–34. doi:`10.1016/j.neucom.2022.05.015`.

Zhakatayev, A., Rakhim, B., Adiyatov, O., Baimyshev, A., Varol, H.A., 2017. Successive linearization based model predictive control of variable stiffness actuated robots, in: IEEE International Conference on Advanced Intelligent Mechatronics (AIM), pp. 1774–1779. doi:`10.1109/AIM.2017.8014275`.

Zhou, T., Droguett, E.L., Mosleh, A., 2022. Physics-informed deep learning: A promising technique for system reliability assessment. Applied Soft Computing 126, 109217. doi:`10.1016/j.asoc.2022.109217`.