Master's thesis

NTNU
Norwegian University of Science and Technology

Jonas Ekeland Kittelsen

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

Master's thesis in Kybernetikk og robotikk
Supervisor: Lars Struen Imsland and Eduardo Camponogara
Co-supervisor: Eric Antonelo
July 2022

NTNU
Norwegian University of
Science and Technology

Jonas Ekeland Kittelsen

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

Master's thesis in Kybernetikk og robotikk
Supervisor: Lars Struen Imsland and Eduardo Camponogara
Co-supervisor: Eric Antonelo
July 2022

Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Summary

Physics-Informed Neural Networks (PINNs) is a method of training a neural network to replicate the behavior of a dynamic system without having access to simulated or measured data, only utilizing the known underlying physical model of the system. Here, this method will be applied to a model of an unstable gas-lifted oil well, utilizing only the known differential equation during training of the PINN. The resulting neural network is able to predict the system's response 60 seconds ahead in time with good accuracy, given the current state measurements and the currently applied control input of the production choke and gas-lift valve. It is also able to predict the system's states further ahead by utilizing the same neural network in a loop, at each iteration progressing 60 seconds ahead in time. This makes the neural network applicable to Model Predictive Control (MPC). The results of this report show several successful MPC experiments controlling both the well states and the bottom-hole pressure of the oil well.

Sammendrag

Physics-Informed Neural Networks (PINNs) er en metode for å trene et nevral nettverk til å gjenskape oppførselen til et dynamisk system uten å ha tilgang til simulerte eller målte data, kun ved å bruke den kjente underliggende fysikken til systemet. Her vil et slikt nett bli trent til å reprodusere dynamikken til en gassløftet oljebrønn, kun ved bruk av systemets differensialligning. Det resulterende nevral nettet er i stand til å predikere systemets respons med god nøyaktighet 60 sekunder frem i tid, gitt den nåværende tilstanden til systemet og den nåværende posisjonen på produksjons ventilen (choken) og gassløft ventilen. Det er også mulig å predikere systemets tilstand lengre frem i tid ved å bruke dette nevral nettet flere ganger i en loop, for hver iterasjons vil vi predikere nye 60 sekunder frem i tid. Dette gjør nevral nettet anvendelig for Model Predictive Control (MPC). Resultatene i denne rapporten viser flere vellykkede MPC eksperimenter som kontrollerer både tilstandene i brønn modellen og trykket i bunnen av brønnen (bottomhole).

Contents

Chapter 1

Introduction

Utilizing neural networks to compute numerical solutions to differential equations was introduced as Physics Informed Neural Network (PINN) in 2017 by Rassi et al. [1, 2]. The basic idea being to train a neural network to reproduce the system's behavior utilizing only the underlying known physics, usually in the form of an Ordinary Differential Equation (ODE) or Differential Algebraic Equation (DAE). This work will focus on the use of these PINNs within optimal control, employing the extended PINN framework introduced by Antonelo et al. [3], called Physics Informed Neural Networks for Control (PINC). Traditional PINNs are trained for a fixed initial condition and do not consider a control input, so that a new PINN has to be trained if either of these changes, which they will continuously during a control application. The PINC framework is extended with the initial condition and control input as inputs to the neural network, allowing it to make predictions for any current state and applied control input. In the original work, this method was successfully applied to control the Van der Pol oscillator and the four tanks system [3], it has also been successfully applied to the control of a robot manipulator [4].

In this work, the PINC framework was applied to a gas lifted oil well model developed by Jahanshahi et al. [5]. This model offered several difficulties when it came to training the PINC, due to the nature of the model. For instance, the use of square root operators in the model equations means that the model may not be defined for arbitrary state values. When doing simulations of realistic scenarios, this is not a problem, as we would never move into these undefined domains. But during the initial phases of training the PINC, the output state predictions can be anything, and it is very likely that some predictions will end up in these undefined regions where we will get a negative number within a square root. This will cause the neural network training to fail, unless some modification is implemented. This problem was avoided by applying a maximum operator within the square roots. The model also contains a highly non-linear friction term that hindered good training performance, which was replaced by a polynomial approximation. These are two examples of adaptions that were made to successfully train the PINC for this

oil well model. Most of the changes are not very specific to this model, and so, may be of great use when training PINCs for other systems as well.

Furthermore, this report will showcase two control applications utilizing the PINC with Model Predictive Control (MPC), one where the states of the system are controlled, and the other where the bottom-hole pressure is controlled. For this second case, an extra neural network was applied to estimate the bottom-hole pressure from the state predictions output by the PINC, as using the systems equations to compute this pressure did not work well with the optimization algorithm, which would crash. These control applications demonstrate that the PINC can successfully be applied to the oil well model, however, the main contribution of this report is the explanation of the different steps taken to successively train the PINC to make good predictions on the model, as these may be of great use when applying the PINC framework to other models.

This report is structured as follows:

- *Chapter 2: Background* - gives an introduction to PINN and the extended PINC framework, the MPC formulation used here and finally presenting the gas-lifted oil well model.
- *Chapter 3: Problem description* - introduces the experiments that were completed: training of the PINC, control of the states and control of the bottom-hole pressure.
- *Chapter 4: implementation* - describes the most important steps in obtaining satisfactory results, including:
    - Changes made to the ODE during training of the PINC in Section 4.1.
    - A method for simplifying the selection of domain for the PINC training data in Section 4.2.1.
    - Demonstrating that a simple change of neural network structure (proposed by [6]) can improve the results, in Section 4.4.
    - The additional neural network employed to estimate the bottom-hole pressure, replacing the system's equation that did not work well with the optimization algorithm, in Section 4.5.
- *Chapter 5: Results* - shows the resulting prediction performance of the PINC along with the two control experiments.
- *Chapter 6: Discussion* - discusses the results in light of the methods that were applied.
- *Chapter 7: Conclusion and further work* - offers a brief conclusion to the discussion, as well as poses interesting topics to be explored further.

Chapter 2

Background

This chapter delivers the background knowledge that is required to understand the remainder of this report. Describing physics informed neural networks, model predictive control, and finally presenting the oil well model.

2.1 Physics Informed Neural Networks

Physics Informed Neural Networks (PINNs) as introduced by [1, 2] is a method for training neural networks to reproduce the solution of a dynamic system by utilizing the known (or partially known) underlying physics of the system. This greatly reduces the need for training data, in some cases removing it completely. How this works was explained thoroughly in the work leading up to this dissertation [7], here only a short introduction to the basics will be given.

The main idea is that we want to train a neural network to reproduce the solution of an ODE. Let us consider the Initial Value Problem (IVP):

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}), \quad \mathbf{y}(0) = \mathbf{y}_0 \tag{2.1}$$

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 initial value problem can be solved analytically, for others, obtaining an analytic solution is not possible or practical, then we have the choice of using numerical methods, for instance the Runge-Kutta method. PINNs offer a different method of solving such problems. The primary purpose of this dissertation is not to solve these kinds of IVP problems, but rather to use an extended version of the PINN framework applied for control purposes, but first we will have a look at the PINN basics.

Figure 2.1: PINN structure for solving an IVP. The neural network maps the time t to the state at this time: $\mathbf{y}(t)$.

Consider the setup illustrated in figure 2.1, a neural network with the time t as input, and the state prediction at this time as output: $\mathbf{y}(t)$. The neural network will be trained to make predictions on the time horizon $t \in [0, T]$. If we want to solve an IVP for the same system, but with different initial condition, we need to train a new neural network, as the initial conditions are "trained" into the weights of the neural network.

2.1.1 Loss function

A PINN network is in essence a normal neural network, the difference lies in the loss function used during training of the neural network. It is composed of two terms:

$$MSE = MSE_y + MSE_F \tag{2.2}$$

The first term, MSE_y is used to impose the initial condition on the neural network, which works a lot like training of traditional neural networks, using input data with corresponding output targets. The second term, MSE_F, however, works differently and is the core idea of PINNs. We will use this term to impose the physics of the system on the neural network output by penalizing the neural networks predictions for not satisfying the ODE. This term is sometimes referred to as a regularization term, as it enters the loss function similarly. But our term is a bit different, it is there to impose a specific solution on the neural network, whereas the goal of traditional regularization terms usually are to penalize large magnitudes of the weights.

Initial conditions

To impose the initial condition on the neural network, we will create a training data point consisting of an input to the neural network and a target for the output of the neural network. The input will be $t = 0$, as it is the initial condition we are interested in. The desired output for $t = 0$ is the initial condition $\hat{\mathbf{y}}$, which will be the target output for the neural network. The following equation shows how the

first term of Equation (2.3) is calculated:

$$MSE_y = \frac{1}{N_y} \|\mathbf{y}(0) - \hat{\mathbf{y}}\|^2 \tag{2.3}$$

where $\mathbf{y}(0)$ is the output of the neural network when inputting $t = 0$, $\hat{\mathbf{y}}$ is the initial condition of the IVP, which is the target for the neural network output. N_y is the dimension of $\mathbf{y}$. $\|\cdot\|$ is the ℓ_2 norm, which is used to compute a measure of the deviation between the output and its target. This deviation is squared and divided by the number of states in the system N_y, to compute the Mean Squared Error (MSE), a common choice of loss function for regression problems within machine learning.

Physics

To impose the physics of the underlying system on the neural network, we want the neural network output to satisfy the ODE for the entire time horizon of interest $t \in [0, T]$. For this, we create input points covering the entire time horizon, these input points are referred to as collocation points. When solving an IVP of an ODE, the only input to the neural network is the time. The collocation points will then be a sequence of times: t^k for $k = 1, \ldots, N_f$, where N_f is the number of collocation points. These points are sampled randomly from the time interval of interest $[0, T]$, ensuring that the interval is covered well.

For each of the collocation points, a forward pass of the neural network is made to calculate the corresponding output $\mathbf{y}(t^k)$. This output is used to calculate the deviation from the ODE in a residual function named $\mathbf{F}$:

$$\mathbf{F}(\mathbf{y}) := \frac{\partial \mathbf{y}}{\partial t} - \mathbf{f}(\mathbf{y}) \tag{2.4}$$

The first term is the neural network output $\mathbf{y}(t)$ differentiated with respect to the time input of the neural network. This derivative is obtained using automatic differentiation, which is available in the library used to implement the neural network (here TensorFlow). The second term of this equation, $\mathbf{f}(\mathbf{y})$, is the ODE calculated for this output. If these two terms match, our neural network's solution satisfies the ODE. This $\mathbf{F}$ is applied to every collocation point, and used to calculate the physics related loss MSE_F:

$$MSE_F = \frac{1}{N_f} \sum_{k=1}^{N_f} \frac{1}{N_y} \left\| \mathbf{F}(\mathbf{y}(t^k)) \right\|^2 \tag{2.5}$$

Again, N_y is the number of states in the system and N_f is the number of collocation points.

We now have the two components of the loss function. The intuition behind the concept is the following; if our neural network is able to 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 neural network is a solution to the IVP.

2.1.2 Training the neural network

For training PINNs it is common to use a combination of the Adam [8] algorithm and the L-BFGS algorithm [1], [2], [3], [4], [9], [10]. The most common approach is to start with a relatively short training using Adam, and then train for as long as necessary with L-BFGS. Usually, L-BFGS is much more efficient for these problems than Adam, but it also seems to be more prone to ending up in a poor local minimum, the initial use of Adam seems to decrease this issue.

2.2 Physics Informed Neural Networks for Control

The main purpose of this dissertation is to apply the PINN framework in the control of dynamic systems. For this, the framework needs to be extended, as proposed in [3]. In the traditional PINN framework, the initial condition is fixed, and it does not support a varying control input. The proposed framework from [3] adds the initial condition and control input as inputs to the neural network, leading to a new framework which will be referred to as PINC (Physics Informed Neural Networks for Control). This increases the input dimension of the neural network, along with the time required for training, but the resulting neural network is able to make predictions from any initial condition and any control input. Again, this is described in detail in the work leading up to this dissertation [7], only a brief introduction will be given here. Figure 2.2 illustrates the concept where the time, initial condition and control input are mapped to a state prediction at the input time.

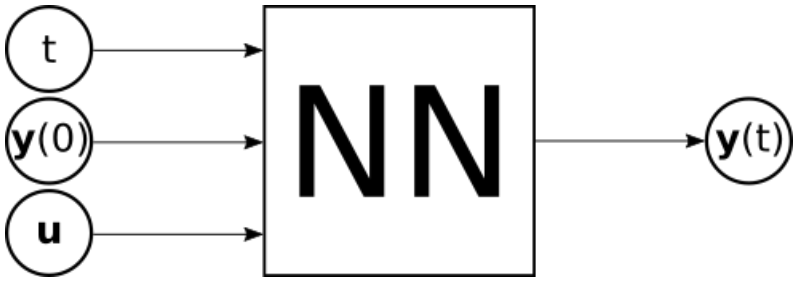


Figure 2.2: Neural network with time t, initial condition $\mathbf{y}(0)$ and control input $\mathbf{u}$ as inputs. This neural network can make predictions of the state at time t given any initial condition and control input. This will be referred to as a PINC model.

Some changes have to be made to the training regime to account for these changes, namely the input data needs to be extended with the extra inputs. We also need to select ranges for the new inputs. For the control input $\mathbf{u}$, this should be the feasible

range of the control input. The range for the initial condition should contain the states of the system we can encounter in operation, so that the neural network is trained to make predictions from any state we may encounter during operation.

2.2.1 Loss function

The essence of the loss function is the same, one component for the initial conditions and one for the physics based loss:

$$MSE = \lambda_y \cdot MSE_y + \lambda_F \cdot MSE_F \tag{2.6}$$

The only difference here is that we have added scaling factors to the two terms, which may also be applied to the traditional PINN, but was left out for the sake of easy understanding. Here, the scaling factors are scalar, 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 become vectors with the same dimension as $\mathbf{y}$. This would allow selecting priority for training of the different states, which may help in obtaining a better result.

Initial conditions

For the traditional PINN only one point was needed to train the initial condition, now however, our initial condition belongs to a range of values, and we also need to consider different control inputs. The training data to impose the initial conditions needs to reflect this. We will create a sequence 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 and is 0 for all the data points because they are for training the initial condition. $\mathbf{y}_0^j$ and $\mathbf{u}^j$ should be sampled randomly for their ranges of interest, and should cover the ranges of values well. The target for each of these training data points is the initial condition that is a part of 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:

$$MSE_y = \frac{1}{N_t} \sum_{j=1}^{N_t} \frac{1}{N_y} \left\| \mathbf{y}(\mathbf{v}^j) - \hat{\mathbf{y}}^j \right\|^2 \tag{2.7}$$

Physics

The collocation points use to impose the ODE on the neural network does also need to be extended with the new inputs. These collocation points are $\mathbf{v}^k, k = 1, \ldots, N_f$, where each contains the input for the neural network $\mathbf{v}^k = \left[t^k, \mathbf{y}_0^k, \mathbf{u}^k\right]^T$. All three elements of the 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:

$$\mathbf{F}(\mathbf{y}) := \frac{\partial \mathbf{y}}{\partial t} - \mathbf{f}(\mathbf{y}, \mathbf{u}) \tag{2.8}$$

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 ($\mathbf{v}^k$):

$$MSE_F = \frac{1}{N_f} \sum_{k=1}^{N_f} \frac{1}{N_y} \left\| \mathbf{F}(\mathbf{y}(\mathbf{v}^k)) \right\|^2 \tag{2.9}$$

2.2.2 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 will use the current state of the system to make a prediction at time T, then we will use this prediction as input to the PINC to obtain a new prediction at time $2T$. We can continue this feeding back of the state, predicting T seconds at every iteration, until we reach the desired prediction time. Figure 2.3 illustrates how this works, feeding the output of the neural network back into the initial condition. Note that, for the very first iteration, the initial condition will have to come from the outside.

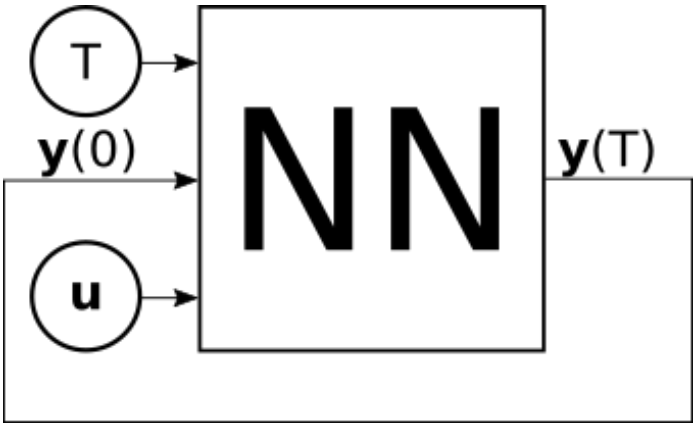


Figure 2.3: PINC in self-loop mode, allowing for long range simulations. The output state prediction at the time $t = T$ is fed back as the initial condition of the PINC to make a new prediction, progressing T seconds every iteration. A different control input may be applied at every iteration. For the very first iteration, the initial condition will have to come from the outside.

When used with MPC, the prediction time T of the PINC will be selected to match the length of a single step in the MPC, so that the PINC with input $t = T$ can be used as a function that maps the states between the time steps of the MPC. With

this setup, the PINC allows for a different control input to be applied at every time step, which is perfect for MPC applications.

2.3 Model Predictive Control

In this work, Model Predictive Control (MPC) will be utilized to control an oil wells. The theory behind MPC will not be explained here, see for instance [11] for an introduction, here we will only have a look at the formulations that will be used in this work. 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 this can be implemented by a state transition matrix, obtained from discretization of the system. For a non-linear system, such as the well model, we can for instance use a linearization of the model or a numerical integration method. Here, we will utilize the PINC as this function that maps the current state and control input to the state at the next time step. As the PINC contain non-linear functions (the activation functions), the resulting MPC will be within the category called Non-linear MPC (NMPC).

Here we will have a look at the formulation used for the MPC to control an oil well:

$$\min \sum_{i=1}^{N} (\mathbf{y}[k] - \mathbf{y}^{ref})^T \mathbf{Q} (\mathbf{y}[k] - \mathbf{y}^{ref}) + \sum_{i=0}^{N_u - 1} \Delta \mathbf{u}[k+i]^T \mathbf{R} \Delta \mathbf{u}[k+i] \tag{2.10a}$$

Subjected to:

$$\mathbf{x}[k+j+1] = \mathbf{F}(\mathbf{x}[k+j], \mathbf{u}[k+j]), \quad j = 0, \ldots, N-1 \tag{2.10b}$$
$$\mathbf{y}[k+j] = \mathbf{F}_{\mathbf{y}}(\mathbf{x}[k+j], \mathbf{u}[k+j-1]), \quad j = 1, \ldots, N \tag{2.10c}$$
$$\mathbf{u}[k+j] = \mathbf{u}[k+j-1] + \Delta \mathbf{u}[k+j], \quad j = 0, \ldots, N_u - 1 \tag{2.10d}$$
$$\mathbf{u}[k+j] = \mathbf{u}[k+N_u-1], \quad j = N_u, \ldots, N-1 \tag{2.10e}$$
$$\mathbf{h}(\mathbf{x}[k+j], \mathbf{y}[k+j], \mathbf{u}[k+j-1]) \leq 0, \quad j = 1, \ldots, N \tag{2.10f}$$

The prediction horizon of the MPC is N steps, and the control input is allowed to change for the first N_u steps. Equation (2.10a) 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. The matrix $\mathbf{R}$ weighs the penalization of the changes in control inputs for the first N_u iterations, after this the control input is fixed. Equation (2.10b) ensures that the solution satisfies the dynamics of the system, here the PINC will be used as the mapping from one state to the next. $\mathbf{F}_{\mathbf{y}}$ in Equation (2.10c) is a function that maps the state and control inputs to some output variable $\mathbf{y}$, which is the variable we wish to control. This will be either the states themselves or the bottom-hole pressure.

The control input is only allowed to change for the first N_u iterations, after this it will be constant and equal to the control input at time step N_u. Equation (2.10d) ensures that the first N_u control inputs change by their respective $\Delta\mathbf{u}$ factor, and Equation (2.10e) ensures that the later outputs are kept constant. Finally, Equation (2.10f) is used to set inequality constraints for the controller. This is used to limit the value of the control inputs $\mathbf{u}$, and can also be used to set limits on states or other variables of interest.

2.3.1 Successive Linearization based MPC (SLMPC)

The NMPC utilizing the PINC will be compared against a Successive Linearization based MPC (SLMPC), which is a controller that at every time step makes a linearization of the model and solves a linear MPC problem. This kind of controller has previously been applied with success to chemical reactors [12] and variable stiffness actuated robots [13].

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:

$$\Delta\mathbf{x}_{k+j+1} = \Delta\mathbf{x}_{k+j} + \mathbf{A}_k\Delta\mathbf{x}_{k+j} + \mathbf{B}_k\Delta\mathbf{u}_{k+j} + \delta_k \tag{2.11a}$$

$$\mathbf{y}_{k+j} = \mathbf{C}_k\Delta\mathbf{x}_{k+j} + \mathbf{D}_k\Delta\mathbf{u}_{k+j} + \mathbf{y}_k \tag{2.11b}$$

$$\Delta\mathbf{x}_{k+j} = \mathbf{x}_{k+j} - \mathbf{x}_k, \quad \Delta\mathbf{u}_{k+j} = \mathbf{u}_{k+j} - \mathbf{u}_k \tag{2.11c}$$

where

$$\mathbf{A}_k = T\frac{\partial\mathbf{f}}{\partial\mathbf{x}_k}(\mathbf{x}_k,\mathbf{u}_k), \quad \mathbf{B}_k = T\frac{\partial\mathbf{f}}{\partial\mathbf{u}_k}(\mathbf{x}_k,\mathbf{u}_k), \quad \delta_k = T\mathbf{f}(\mathbf{x}_k,\mathbf{u}_k) \tag{2.12a}$$

$$\mathbf{C}_k = \frac{\partial\mathbf{h}}{\partial\mathbf{x}_k}(\mathbf{x}_k,\mathbf{u}_k), \quad \mathbf{D}_k = \frac{\partial\mathbf{h}}{\partial\mathbf{u}_k}(\mathbf{x}_k,\mathbf{u}_k), \quad \mathbf{y}_k = \mathbf{h}(\mathbf{x}_k,\mathbf{u}_k) \tag{2.12b}$$

where T is the discretization time step length, $\mathbf{f}(\mathbf{x},\mathbf{u})$ is the system ODE, and $\mathbf{h}$ is a function that computes the output from the states and control input: $\mathbf{y} = \mathbf{h}(\mathbf{x},\mathbf{u})$. The linear model in Equation (2.11a) can then be used in place of the PINC in the constraint of Equation (2.10b), and the linearized output Equation (2.11b) is used to implement the constraint in Equation (2.10c).

2.4 Oil well model

The plant model used here is of a gas-lifted oil well, which was developed by Jahanshahi, Skogestad and Hansen in [5]. Below the model is stated and explained briefly, for an in depth derivation of the model see [5].

During the model derivation in [5], some simplifications are made in the algebraic equations to avoid implicit terms. These simplifications result in the model being

explicit, which means that there is no need for a Differential Algebraic Equation (DAE) solver, thus the dynamic system can be treated as an ODE and simulated using for instance the Runge-Kutta method. These simplifications are made in the equations for velocities, and will be explained further along these equations.

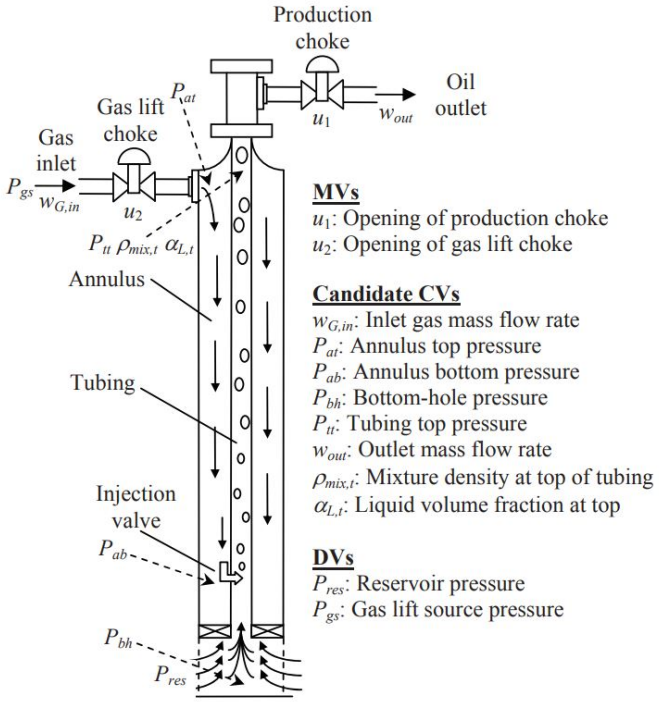


Figure 2.4: Structure of the oil well. Source: Jahanshahi, Skogestad and Hansen in [5]

Figure 2.4 shows an illustration of the well. Liquid and gas from the reservoir enters the well at the bottom of the tubing, referred to as the bottom-hole. From there, the gas and liquid flows up through the tubing, until they exit through the production choke. The tubing is surrounded by a bigger 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) located near the bottom of the annulus. This gas helps "lifting" 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 not sufficient to sustain the flow from the reservoir to the top side.

There are a lot of variables in the model, many of the same nature (pressures, densities, etc.) but for different locations of the well. Subscript abbreviations are used extensively in the variables to refer to the phase (gas or liquid) and location. The variables are listed in Table 2.1. An important remark is that this model does not separate between water and oil, it only considers a liquid phase.

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

Table 2.1: Abbreviations used in the subscript of the model variables to indicate the phase and location which the variable represents.

2.4.1 States

The model consists of three states: mass of gas in the annulus $m_{G,an}$, mass of gas in the tubing $m_{G,tb}$ and mass of liquid in the tubing $m_{L,tb}$. The differential equations are:

$$\dot{m}_{G,an} = w_{G,in} - w_{G,inj} \tag{2.13a}$$
$$\dot{m}_{G,tb} = w_{G,inj} + w_{G,res} - w_{G,out} \tag{2.13b}$$
$$\dot{m}_{L,tb} = w_{L,res} - w_{L,out} \tag{2.13c}$$

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

The equations used to calculate these flows follows here, few of them will be explained, see [5] for more details.

2.4.2 Mass flows

$$w_{G,in} = K_{gs} u_2 \sqrt{\rho_{G,in} max(P_{gs} - P_{at}, 0)} \tag{2.14a}$$
$$w_{G,inj} = K_{inj} \sqrt{\rho_{G,an,b} max(P_{an,b} - P_{tb,b}, 0)} \tag{2.14b}$$
$$w_{res} = PI max(P_{res} - P_{bh}, 0) \tag{2.14c}$$
$$w_{L,res} = (1 - \alpha^m_{G,tb,b}) w_{res} \tag{2.14d}$$
$$w_{G,res} = \alpha^m_{G,tb,b} w_{res} \tag{2.14e}$$
$$w_{out} = K_{pr} u_1 \sqrt{\rho_{mix,tb,t} max(P_{tb,t} - P_0), 0} \tag{2.14f}$$
$$w_{L,out} = (1 - \alpha^m_{G,tb,t}) w_{out} \tag{2.14g}$$
$$w_{G,out} = \alpha^m_{G,tb,t} w_{out} \tag{2.14h}$$

2.4.3 Pressures

$$P_{an,t} = \frac{RT_a m_{G,an}}{M_G V_a} \tag{2.15a}$$

$$P_{an,b} = P_{an,t} + \frac{m_{G,an} g L_{an}}{V_{an}} \tag{2.15b}$$

$$P_{tb,t} = \frac{\rho_{G,tb,t} R T_{tb}}{M_G} \tag{2.15c}$$

$$P_{tb,b} = P_{tb,t} + \overline{\rho}_{mix} g L_{tb} + F_{tb} \tag{2.15d}$$

$$P_{bh} = P_{tb,b} + F_{bh} + \rho_L g L_{bh} \tag{2.15e}$$

2.4.4 Densities

$$\rho_{G,an,b} = \frac{P_{an,b} M_G}{RT_{an}} \tag{2.16a}$$

$$\rho_{G,in} = \frac{P_{gs} M_G}{RT_{an}} \tag{2.16b}$$

$$\rho_{G,tb,t} = \frac{m_{G,tb}}{V_{tb} + S_{bh} L_{bh} - m_{L,tb}/\rho_L} \tag{2.16c}$$

$$\overline{\rho}_{mix} = \frac{m_{G,tb} + m_{L,tb} - \rho_L S_{bh} L_{bh}}{V_{tb}} \tag{2.16d}$$

$$\rho_{G,tb,b} = \frac{P_{tb,b} M_G}{RT_{tb}} \tag{2.16e}$$

$$\rho_{mix,tb,t} = \alpha_{L,tb,t} \rho_L + (1 - \alpha_{L,tb,t}) \rho_{G,tb,t} \tag{2.16f}$$

2.4.5 Mass/liquid fractions

$$\overline{\alpha}_{L,tb} = \frac{m_{L,tb} - \rho_L S_{bh} L_{bh}}{V_{tb} \rho_L} \tag{2.17a}$$

$$\alpha^m_{G,bh} = GOR/(GOR + 1) \tag{2.17b}$$

$$\alpha_{L,tb,b} = \frac{w_{L,res} \rho_{G,tb,b}}{w_{L,res} \rho_{G,tb,b} + (w_{G,inj} + w_{G,res}) \rho_L} \tag{2.17c}$$

$$\alpha_{L,tb,t} = 2\overline{\alpha}_{L,tb} - \alpha_{L,tb,b} \tag{2.17d}$$

$$\alpha^m_{G,tb,t} = \frac{(1 - \alpha_{L,tb,t}) \rho_{G,tb,t}}{\alpha_{L,tb,t} \rho_L + (1 - \alpha_{L,tb,t}) \rho_{G,tb,t}} \tag{2.17e}$$

2.4.6 Velocities

$$\overline{U}_{L,tb} = \frac{4(1-\alpha^m_{G,bh})\overline{w}_{res}}{\rho_L \pi D^2_{tb}} \tag{2.18a}$$

$$\overline{U}_{G,tb} = \frac{4(w_{G,in} + \alpha^m_{G,bh}\overline{w}_{res})}{\rho_{G,tb,t} \pi D^2_{tb}} \tag{2.18b}$$

$$\overline{U}_{mix,tb} = \overline{U}_{L,tb} + \overline{U}_{G,tb} \tag{2.18c}$$

$$\overline{U}_{L,bh} = \frac{\overline{w}_{res}}{\rho_L S_{bh}} \tag{2.18d}$$

These equations express the liquid, gas and mix velocities in the tubing, and the liquid velocity in the bottom-hole. Equations (2.18a), (2.18b) and (2.18d) use the predefined constant $\overline{w}_{res}$ as the flow from the reservoir, which is one of the simplifications made in the model derivation in [5] to avoid an implicit set of equations.

Equation (2.18b) also has another simplification, again to avoid an implicit set of equations, $w_{G,inj}$ is replaced by $w_{G,in}$. That means that when calculating the average velocity of the gas in the tubing, we do not use the mass flow of gas injected into the tubing, but instead the gas mass flow injected into the annulus through the gas-lift choke. With this substitution, a change in gas-lift choke opening will directly influence the velocities in the tubing, which will in turn affect the tubing friction loss F_{tb} in Equation (2.19c), which affects the pressure in the bottom of the tubing $P_{tb,b}$. This change happens much faster than the change in pressure at the bottom of the annulus $P_{an,b}$, which slowly increases (decreases) as the gas flow into the annulus through the gas-lift choke is increased (decreased). As can be seen from Equation (2.14b), $w_{G,inj}$ is dependent on the differential pressure between the bottom of the annulus and the bottom of the tubing. The bottom line is that this simplification results in that a change in gas-lift choke opening has an immediate effect on the gas injected into the tubing, $w_{G,inj}$. In the physical system this would be a slower process as the pressure builds up at the bottom of the annulus.

2.4.7 Friction terms

$$Re_{tb} = \frac{\overline{\rho}_{mix,tb}\overline{U}_{mix,tb}D_{tb}}{\mu} \tag{2.19a}$$

$$\frac{1}{\sqrt{\lambda_{tb}}} = -1.8\log_{10}\left[\left(\frac{\epsilon/D_{tb}}{3.7}\right)^{1.11} + \frac{6.9}{Re_{tb}}\right] \tag{2.19b}$$

$$F_{tb} = \frac{\overline{\alpha}_{L,tb}\lambda_{tb}\overline{\rho}_{mix,tb}\overline{U}^2_{mix,tb}L_{tb}}{2D_{tb}} \tag{2.19c}$$

$$Re_{bh} = \frac{\rho_L\overline{U}_{L,bh}D_{bh}}{\mu} \tag{2.19d}$$

$$\frac{1}{\sqrt{\lambda_{bh}}} = -1.8\log_{10}\left[\left(\frac{\epsilon/D_{bh}}{3.7}\right)^{1.11} + \frac{6.9}{Re_{bh}}\right] \tag{2.19e}$$

$$F_{bh} = \frac{\lambda_{bh}\rho_L\overline{U}^2_{L,bh}L_{bh}}{2D_{bh}} \tag{2.19f}$$

These equations calculate the friction pressure-loss terms for the tubing (2.19c) and the bottom-hole (2.19f). For the bottom-hole friction calculations, the average mass flow from the reservoir, $\overline{w}_{res}$, is used, as can be seen in Equation (2.18d), resulting in Equations (2.19d), (2.19e) and (2.19f) being constant, which is another of the simplifications made in the derivation of the model to make it explicit.

2.4.8 Parameters of the model

Table 2.2 shows the model parameters used for the primary oil well, which were extracted from [5], except for ϵ which could not be found in the article, instead it was collected from [14]. Several oil wells were constructed by altering the parameters of the model, their parameters are shown in Table 2.3.

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}	$\mathrm{Pa \cdot s}$
ρ_L	liquid density	760	$\mathrm{kg/m^3}$
M_G	gas molecular weight	0.0167	kg/mol
T_{an}	annulus temperature	348	K
V_{an}	annulus volume	64.34	$\mathrm{m^3}$
L_{an}	annulus length	2048	$\mathrm{m^3}$
P_{gs}	gas source pressure	140	bar
S_{bh}	cross-section below injection point	0.0314	$\mathrm{m^2}$
L_{bh}	length below injection point	75	m
T_{tb}	injection point tubing temperature	369.4	K
GOR	mass gas oil ratio	0	—
P_{res}	reservoir pressure	160	bar
$\overline{w}_{res}$	average mass flow from reservoir	18	kg/s
D_{tb}	tubing diameter	0.134	m
L_{tb}	tubing length	2048	m
V_{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_{gs}	gas-lift choke cons.	9.98×10^{-5}	—
K_{inj}	injection valve cons.	1.40×10^{-4}	—
K_{pr}	production choke cons.	2.90×10^{-3}	—

Table 2.2: Parameters for the oil well model from [5] (ϵ from [14]). These are the parameters that will be used for the first analysis in this work.

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 2.3: Parameters for all the wells.

Chapter 3

Problem description

The main goal of this work is to implement NMPC controlling the bottom-hole pressure of the gas-lifted oil well, but on the way to this goal, there will be several intermediate stages with their own analysis and results. These main stages will be introduced in this chapter, and will be analyzed and discussed later in the dissertation.

As the current framework for the PINC is to have the current states as input, we will assume for all the MPC application that we know the initial conditions, and that, at every time step, we measure or estimate the current system state. As all the states are masses of gas or liquid in the well, it is not realistic that we can measure these directly, possibly an observer could be added to estimate these states or some modifications could be made to the PINC framework, however this is outside the scope of this project.

The main purpose of this work is to showcase the control of the oil well using NMPC with the PINC. However, all control experiments will be compared with the SLMPC.

3.1 Modeling the oil well using PINC

The first step will be to create a PINC network that can make accurate predictions on the oil well system. This PINC network will be trained without the use of any measurement data, only utilizing the ODE. One PINC network will be trained for each of the oil wells to be used in the final control problem.

To evaluate the performance of the PINCs predictions, it will be put in self-loop mode with random initial conditions and a random sequence of control inputs, which will be compared 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 real 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:

$$IAE = \frac{1}{C}\sum_{k=1}^{C}\frac{1}{N_y}\|\mathbf{y}[k]-\mathbf{r}[k]\| \tag{3.1}$$

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.

3.2 MPC of a singe well with state reference

The first application of the oil well PINC within control will be to use MPC to control the states of the oil well. This is not a very practical application, but offers an intermediate step on the way to controlling the bottom-hole pressure. The PINC network will be utilized as the mapping between the states at different time steps, namely $\mathbf{F}$ of Equation (2.10b) in the MPC formulation from Section 2.3. Further, the output we wish to control is simply the states, $\mathbf{y}[k] = \mathbf{x}[k]$.

To display the performance of the controller, some step changes will be made to the reference. The response will be analyzed visually and in terms of IAE, where $\mathbf{y}[k]$ of Equation (3.1) is the system's state and $\mathbf{r}[k]$ is the corresponding reference.

3.3 MPC of a single well with bottom-hole pressure reference

A more realistic scenario is that we want to control the bottom-hole pressure, which is the variable that determines the flow from the reservoir into the well (equation 2.14c). As the PINC outputs only the state predictions, some method will have to be utilized to compute the bottom-hole pressure from the states: for time k the bottom-hole pressure prediction is calculated as $y[k+j] = F_y(\mathbf{x}[k+j], \mathbf{u}[k+j-1])$.

Again, the performance will be assessed visual and using the IAE during step changes to the bottom-hole pressure reference.

Chapter 4

Implementation

Most of the concepts described in chapter 2 are not straight forward to implement. In this chapter, the most important implementation details will be presented. Some of these details were not apparent at first glance and took a long time to figure out, so this chapter may contain tips of great value for who is working on similar problems, especially for training of PINC networks.

4.1 Changes to the model equations during training of the neural network

As the ODE equations are used in the loss function of the neural network, they have to be differentiated during training of the neural network to obtain the gradients of the loss term. Some of these equations can cause problems during training, so some modifications have to be made for the neural network to train reliably. The changes that were made are described in this section. Keep in mind that these changes are only made to the ODE used to calculate MSE_F. Thus, the original equations are still used for all simulations in this report, which will represent the real system.

4.1.1 Tubing friction factor

Equation (2.19b) is highly non-linear and is not defined for negative numbers inside the logarithm function. For the argument of the logarithm to be negative, the Reynolds number in the tubing must be sufficiently negative, which is not physically possible. But during the initial phase of training the neural network, it may output predictions resulting in a negative Reynolds number.

The complex non-linearity of this equation seems to result in slow training convergence. To improve on both of these issues, we replace the equation with an approximation using a third order polynomial in the region of interest of Reyn-

olds numbers, which can be found through test simulations. Figure 4.1 shows an example of this approximation for the first well. 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.

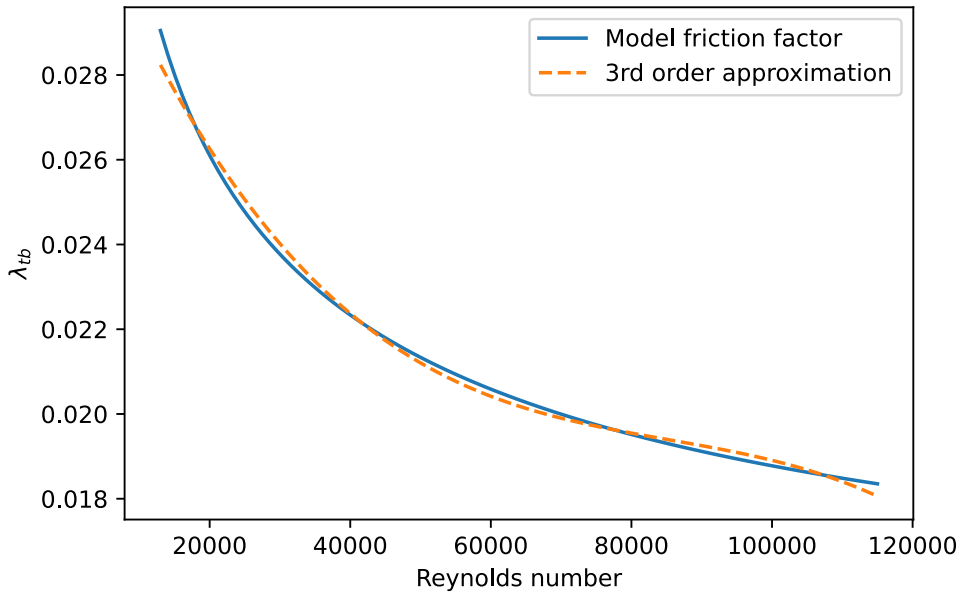


Figure 4.1: The third order polynomial approximation of the tubing friction factor in Equation (2.19c) in orange compared with the real friction factor in blue. This plot shows the approximation for the first well, which was fitted on the interval $[13000, 115000]$ using the `curve_fit` function of SciPy.

During the initial phase of training, the neural network will output poor predictions that can be far outside physically feasible values. We can then end up with Reynolds numbers outside the interval for which the approximation was fit. The simplest tweak is to truncate the Reynolds number (calculated from the PINC prediction) to the selected interval. As this removes the dependence on the preceding variables, it also reduces the quality of the gradients when the Reynolds number gets truncated, but in practice this works just fine. Equation (4.1) shows this approximation with the truncation of Reynolds number, with $g(\cdot)$ being the third order polynomial approximation shown in Equation (4.2).

$$\lambda_{tb}(Re_{tb}) = \begin{cases} g(Re_{tb,min}), & Re_{tb} < Re_{tb,min} \\ g(Re_{tb}), & Re_{tb,min} \leq Re_{tb} \leq Re_{tb,max} \\ g(Re_{tb,max}), & ,Re_{tb} > Re_{tb,max} \end{cases} \tag{4.1}$$

$$g(x) = ax^3 + bx^2 + cx + d \tag{4.2}$$

The numerical values for these coefficients and the ranges of Reynolds numbers used to generate these approximations are shown in table 4.1.

	a	b	c	d	$Re_{tb,min}$	$Re_{tb,max}$
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}		

Table 4.1: Table presenting the numerical values for the 3rd 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 element of the column should be multiplied with, for example, for the first well: $a = -1.78 \cdot 10^{-17}$.

The oil well model also contains a friction factor for the bottom-hole friction, utilizing the same equations. But since the flow from the reservoir is considered constant when calculating the Reynolds number here, this friction factor is also constant, and no changes are needed.

4.1.2 Square root

The square root is another function that can cause numerical problems during training of the neural network, 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 then propagate through the calculations, eventually ending up in the loss function. Then the gradients of the loss function are also "NaN", which are updated into the weights and biases of the neural network, resulting in a useless neural network with undefined weights and biases. This is only a problem during the initial phase of the training, when the neural network may output pretty much anything. When the predictions of the network are approaching their desired values, the outputs will hopefully be in a range where we do not encounter negative values within these functions, but this depends on the selection of training data, which we will discuss soon.

Several approaches were tried to extend the square root for negative values, while maintaining gradients that would move the training in the right direction. In the

end, this turned out to not make a big difference, simply applying the max(·) operator within these functions to avoid negative values was sufficient. Even though this results in no gradients for negative numbers, the training worked out fine. To avoid zero division error in other equations, it is helpful to set the lower limit of the square root input to some small positive value, in the scenarios considered 10^{-3} worked well. This value was found by trial and error, aiming for as small as possible values without creating numerical issues. This modified square root function is:

$$f_{sqrt}(x) = \sqrt{\max\{x, 10^{-3}\}} \tag{4.3}$$

4.2 Single well PINC

Here, the most important steps of designing and training the PINC for the oil well model are described.

4.2.1 Domain of training data

An important point when training a PINC is to select the domain in which we draw our data points for training the neural network. The range for the time input t should be the prediction time we are interested in which, for the case of MPC, is the time of a single step of the MPC, here 60 seconds. For the control input, the ranges should be the range which is feasible for the valve openings. When it comes to the states, this selection is not so obvious. It should include all possible states we can encounter during operation of the system, so that the resulting neural network can make good predictions for any operating condition. Selecting too large an interval will most likely result in reduced performance, so this is not preferable either.

With this model there is another problem when selecting the domain for the states, we may run into states where the model is not define, where a value within a square root is negative. It does not make sense to train the neural network to make predictions on these infeasible states, as we will never encounter them during operation. It was also observed that training on these infeasible areas of the state space greatly reduces training performance of the PINC, reducing the overall prediction performance.

The range for the states was selected by doing several Runge-Kutta simulations with different control inputs, and analyzing what ranges the states reached. The goal here was to find the state values that we may encounter during operation. When only applying lower and higher limits to the states, the resulting domain of the training data is a cube in the three dimension state space, which may be suboptimal, as some areas within this cube may be infeasible. Instead of constructing a more advance shape for drawing training data, a different solution was applied. First, training data point for the initial conditions and collocation points were drawn randomly from within these limits (the cube). Then every point was run

through the ODE to check if this point was feasible. Then all the infeasible points were excluded from the training data. For the first well model, this filter did not offer a noteworthy improvement, as the desired interval for state values included very few of these infeasible points. However, for the second well, removing these infeasible points was required to obtain a decent performance, as it was impossible to construct a large enough interval for the states without running into these infeasible regions.

For the initial condition this is a perfect solution, as we know that if the input is feasible the desired output will also be feasible, as they are the same. However, for the collocation points, this only verifies that the initial condition is feasible, and not that the desired prediction exists. For these points, the desired output is a prediction up to 60 seconds ahead in time. 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.

4.2.2 Model validation

To calculate a validation loss for the model, 100 sets of random initial conditions $\mathbf{y}^i_{0,val}$ and control inputs $\mathbf{u}^i_{val}$ were drawn for within their respective ranges and simulated for one time step of 60 seconds with Runge-Kutta. Any simulation failing due to a negative value within a square root was excluded, and a new point was drawn until 100 valid simulations of 60 seconds were obtained. To calculate the validation error of the model, all these validation points are passed through the neural network along with the time $T = 60$, the output prediction is then compared with the target, obtained by Runge-Kutta simulation $\hat{\mathbf{y}}^i_{val}$. The validation error is calculated as the MSE between the neural network output and the Runge-Kutta simulation:

$$MSE_{val} = \frac{1}{N_{val}N_y}\sum_{i=1}^{N_{val}}\left\|\mathbf{N}(T,\mathbf{y}^i_{0,val},\mathbf{u}^i_{val}) - \hat{\mathbf{y}}^i_{val}\right\|^2 \tag{4.4}$$

where N_{val} is the number of validation simulations and N_y is the number of states of the model. $\mathbf{N}(T,\mathbf{y}^i_{0,val},\mathbf{u}^i_{val})$ is the neural network prediction 60 seconds ahead in time for validation point i, it is compared with $\hat{\mathbf{y}}^i_{val}$, the target output of this validation point. $\|\cdot\|$ is the ℓ_2-norm, here it is squared to obtain the squared error.

This validation method does not put the PINC in self-loop mode, and does therefore not measure the accumulated error during self-looping when doing long range simulation. Applying self-looping during validation calculation would be preferable, because in practice the network will be used in self-loop with MPC to predict over the entire MPC horizon, so this would be a better measure of the performance in operation. But due to the issue with running into negative values in square roots, the choice was made to not utilize self-looping, as these simulations would quickly run into infeasible state values. This measure is also related to the

loss in self-loop; if the neural network can make good predictions from any initial condition (within the selected domain), it should perform well in self-loop as long as the states stay within their selected domain.

4.2.3 Hyperparameter search

This section will describe the steps that were taken to obtain the hyperparameters for the PINC model, including the learning rate for the Adam algorithm, number of training data points, and the number of layers and neurons of the neural network. All the tests here were preformed with a neural network structure containing something that resembles skip connections, which will be introduced in section 4.4. These steps ensured that the performance of the resulting PINC is good and repeatable. During the hyperparameter searches, all different configurations were trained 10 times with different initialization of the neural network weights and biases to get a better picture of the average performance of each setup. For evaluation of the performance, the error on the validation set was used.

Adam learning rate

All the PINCs trained for the hyperparameter search will initially be trained with $1,000$ iterations of the Adam algorithm, no search was conducted to find this number of iterations. The first search was conducted to find a good choice of learning rate for the Adam algorithm. For this search, a PINC with 5 hidden layers consisting of 20 neurons each was trained with a variety of different learning rates: $[1e-4, 4e-4, 7e-4, 1e-3, 4e-3, 7e-3, 1e-2, 4e-2]$. The learning rate $7e-3$ was selected as the best choice.

Number of training data points

The next experiment was conducted to find a good choice for the number of training data points for the initial condition, N_t, and for collocation points, N_f. The same network structure of 5 hidden layers of 20 neurons each was used for this experiment. Again, 10 networks were trained for each combination of training data points, with different weight and bias initialization. All networks were trained initially with $1,000$ epochs of Adam and then $1,000$ iterations of L-BFGS. Figure 4.2 shows plots of the validation losses of these models for different combinations of N_t and N_f. The y-axis has the same limits in all the plots. Some training experiments terminate early due to converging to a poor local minimum. Taking into consideration both the final validation loss and the number of training trails that fail, the combination $N_t = 1,000$ and $N_f = 10,000$ was selected as the best choice.

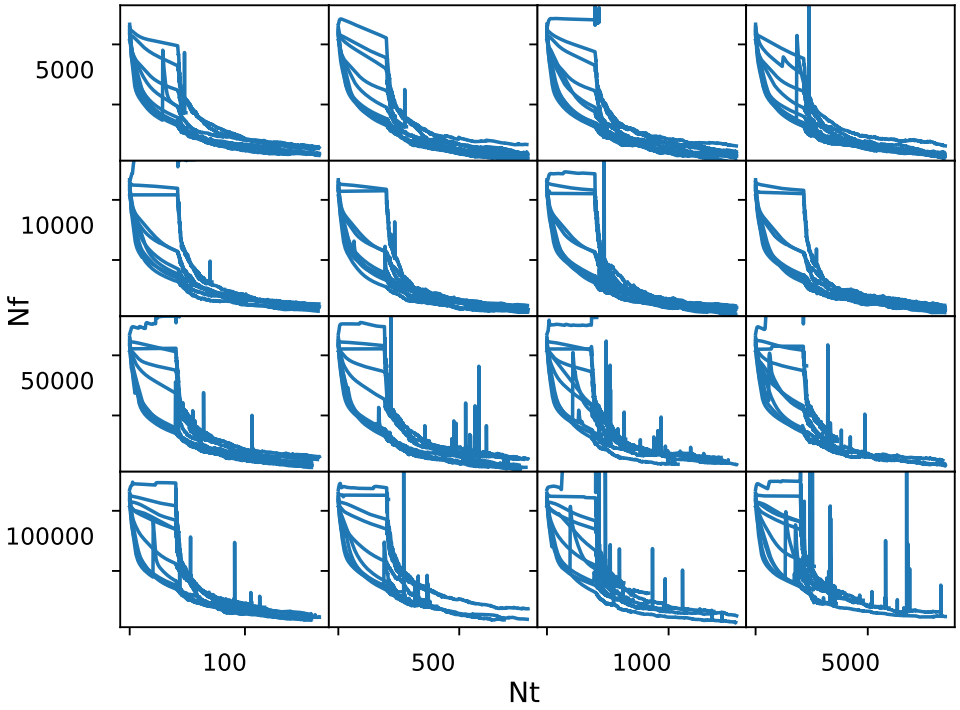


Figure 4.2: Showing the validation loss progression during training of neural networks with different combinations of training data points. N_t is the number of data points used for the initial condition, and N_f is the number of data points used to train the physics related loss. In every subplot, there are ten lines, representing ten neural networks with different weight initialization. The limits on the y-axis is the same for all the subplots. The combination of $N_t = 1,000$ and $N_f = 10,000$ was selected as the best choice, because it had the best average performance, and none of the training attempts terminated early.

4.2.4 Neural network structure

The last experiment was carried out to find the structure of the neural network, the number of hidden layers and number of neurons for each of these layers. Again, 10 neural networks were trained for each combination of layers and neurons for $1,000$ epochs of Adam and $1,000$ iterations of L-BFGS. The result was evaluated based on the number of models that failed to train, and the resulting loss on the validation data for the models that trained successfully. Figure 4.3 shows the validation loss during training for these models, the y-axis has the same limits for all the plots.

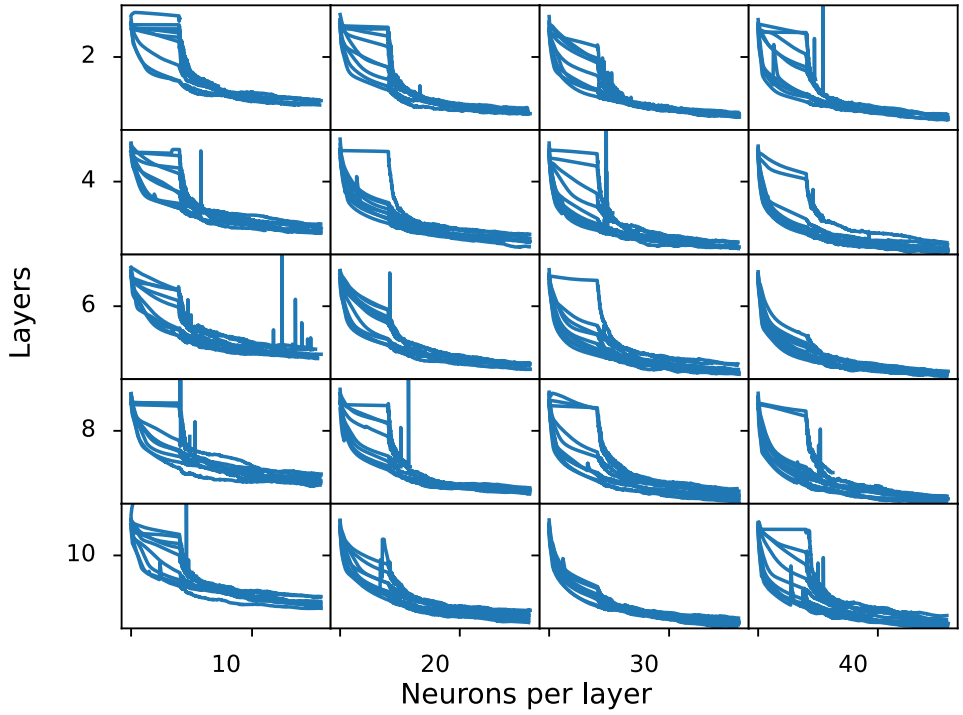


Figure 4.3: Validation error during training of neural networks with different configurations of hidden layers and neurons per layer. Ten neural networks were trained for each configuration, starting with 1,000 epochs of Adam, then 1,000 iterations of L-BFGS. The y-axis has the same limits in all the subplots.

This training regime is quite short, so the training does not converge. For this reason, three of the combinations were selected for an additional $1,000$ iterations of L-BFGS. Based on Figure 4.3 the structures of $6 \cdot 30$, $6 \cdot 40$ and $10 \cdot 30$ layers and neurons were selected for further training. After this additional training, the structure of 6 hidden layers of 30 neurons each was selected as it had the lowest average validation loss.

4.3 Gradients of the physics loss term

Now, we will have a look at an implementation detail of the physics related loss term, MSE_F, that may not be apparent at a first glance. The convergence of the training is highly dependent on the quality of the gradients, so it is worth spending some time investigating this. Recall the residual function $\mathbf{F}$ used in the training of PINNs which is used to measure how well the neural networks output satisfy the ODE:

$$\mathbf{F}(\mathbf{y}) := \frac{\partial \mathbf{y}}{\partial t} - \mathbf{f}(\mathbf{y}, \mathbf{u}) \tag{4.5}$$

where the first term is the output of the neural network differentiated with respect to time, and the second term is the ODE calculated using the neural network output and the control input. At first glance, it may resemble the internal part of the MSE_y loss of Equation (2.7), where the first term accounts for the behavior of the neural network, and the second term is the target. The difference here is that also the second term, $\mathbf{f}(\mathbf{y}, \mathbf{u})$, is a function of the neural network output. This allows us to utilize both terms of Equation (4.5) when calculating the gradients of the weights and biases during training. For simple ODEs this is trivial, but for more complex systems, like the well model, we can run into some problems. These problems can include losing the dependence on the inputs through a maximum operator (in the valve equations), highly non-linear functions, and the introduction of maximum operators inside square roots, as describe earlier.

The way TensorFlow (and also PyTorch) calculates the gradients is through a process called automatic differentiation. This works by creating a computation graph of all the mathematical operations, and differentiating step wise utilizing the chain rule. This allows us to choose whether we want to use both or only one of the terms in Equation (4.5) for computing the gradients. If we want to avoid computing the gradients for one of them, we can avoid tracing the computations in the graph, effectively removing the dependence on the weights and biases when it comes to calculating the gradients.

It turns out that the gradients of second term of the residual $\mathbf{F}$ is very important for training the neural network. Figure 4.4 demonstrates this for a PINN with six layers that is partially trained. The figure is a Kernel Density Estimate (KDE) plot, which can be interpreted similarly to a histogram. It shows the distribution of the gradients for the weights and biases in all the layers. All the plots are centered around zero, so a large spike in the center of the curve means that a lot of the gradients are close to zero, which is undesirable. The blue line shows the KDE of the gradients when utilizing the derivatives of both terms in Equation (4.5), as this curve is lower in the middle, and flatter, it has better gradients than the two others. The orange curve shows the KDE of the gradients when only constructing the computational graph for the first term in Equation (4.5), $\frac{\partial \mathbf{y}}{\partial t}$, then the second term acts as a constant without any derivatives with respect to the weights and biases. The green line shows the KDE of the gradients when only constructing the computational graph for $\mathbf{f}(\mathbf{y}, \mathbf{u})$.

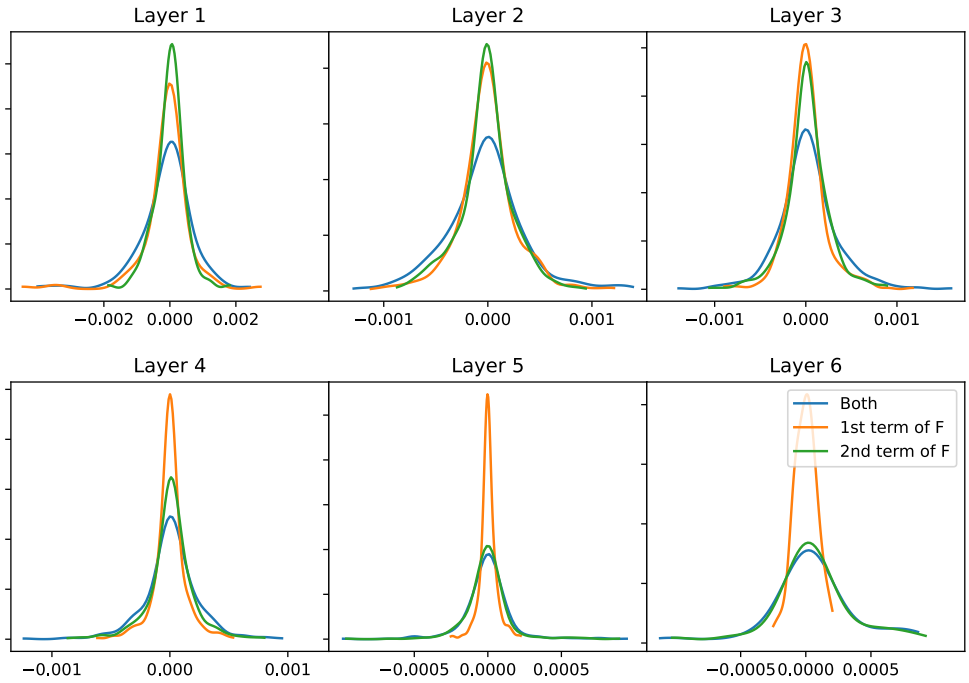


Figure 4.4: Kernel Density Estimate (KDE) of the gradients values for each layer of a partially trained PINN on the oil well model. It can be interpreted as a histogram, for each layer showing the distribution of gradient values for the weights and biases. All plots are centered around zero, so a large spike in the middle indicates that a lot of the gradients are close to zeros, which is undesirable. The orange line shows the KDE for the first term of the residual function **F**, and the green line shows the KDE for the second term. The blue line shows the KDE for both terms summed together. For layer 1-4, we can see that utilizing both terms results in better gradients, as the blue line is clearly better than the others. However, for layer 5 and 6, the 1st term (orange line) does not seem to contribute much, as the green and blue lines are very close.

In the case of Figure 4.4, we can see that the second term of the residual **F** is clearly more important than the first term, as it is flatter, indicating greater absolute values for the gradients. This is especially true for layers 4-6, where the blue and green line nearly match, indicating that the 1st term (orange line) contributes very little to the total gradients (blue). Further, it is clear that combining them yield the best results looking only at the condition of the gradients, not taking computational cost into account.

Figure 4.5 shows a KDE plot of the same neural network, now trained for a longer period. The difference is even greater here. The blue and green curves are very similar, indicating that the second term of the residual **F** accounts for most of the gradients of the residual function, whereas the effect of the first term seems to be negligible. This shows the importance of retaining the computational graph through the ODE. If we were not able to make a differentiable ODE, we would

only have the 1st term, that is: only the orange line in the plot.

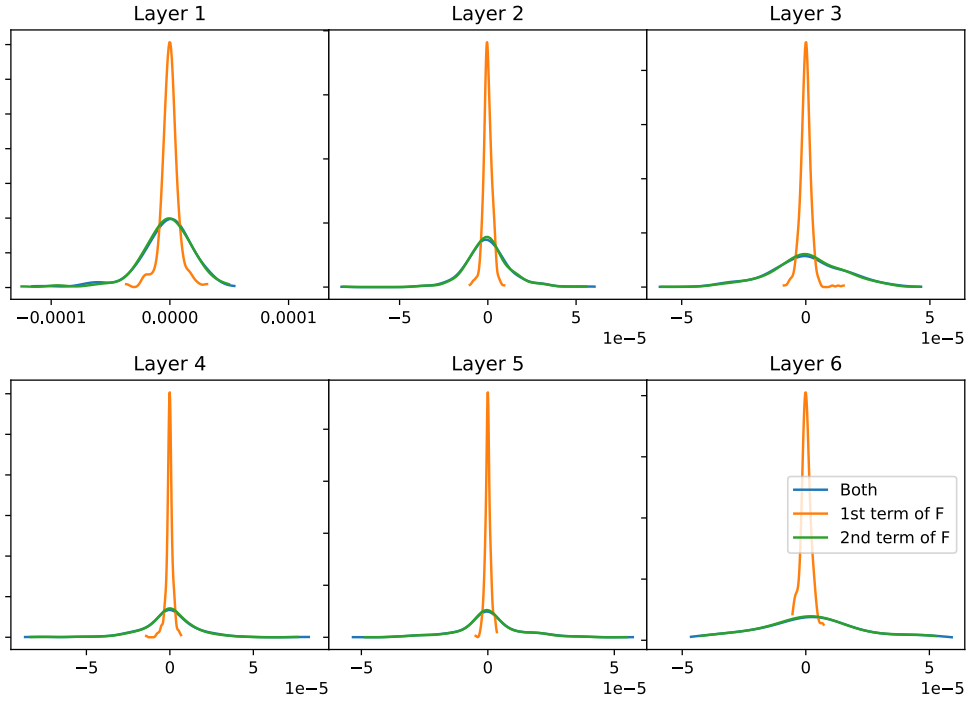


Figure 4.5: KDE of the same PINC as in Figure 4.4, now trained for a longer time. The larger spike in the orange curve indicates that the magnitudes of the gradients are smaller for the 1st term than the 2nd term of the residual function. Clearly, the 2nd term is the most important here.

This is a general trend, at lest for the oil well model. The gradients of the 1st term deteriorate quickly during training. This could be because of the long chain of calculations needed to obtain these gradients; first a forward pass is made to calculate the output $\mathbf{y}$, then we have to go all the way back to the input t to calculate the derivative $\frac{\partial \mathbf{y}}{\partial t}$, then a pass forward to calculate the gradients of the weights and biases.

4.4 An improved neural network structure

From the last section, it is clear that poor gradients can be an issue when training the neural network to predict on this model. Anything we can do to improve the magnitude of the gradients can be of great help in training the neural network. One approach is to use a different structure in the neural network to avoid vanishing gradients, for instance skip connections. One such change of network structure that resembles skip connections is proposed by [6]. In addition to the fully connected dense network, this structure adds two more layers, called encoders. The

input to both these encoder layers are the neural network input, while the output of both these encoders are used to calculate the activation of each layer in the main fully connected network, except for the final layer. This ensures that the output of the neural network is closer connected to the input and the trainable variables. The mathematical formulation of this proposed structure is shown in Equation (4.6) and a visual representation is shown in figure 4.6:

$$U = \phi\left(W^1 X + b^1\right), \quad V = \phi\left(W^2 X + b^2\right) \tag{4.6a}$$

$$Z^{(1)} = \phi\left(W^{z,1} X + b^{z,1}\right) \tag{4.6b}$$

$$A^{(1)} = \left(1 - Z^{(1)}\right) \odot U + Z^{(1)} \odot V \tag{4.6c}$$

$$Z^{(k)} = \phi\left(W^{z,k} A^{(k-1)} + b^{z,k}\right), \quad k = 2, \ldots, N_L \tag{4.6d}$$

$$A^{(k)} = \left(1 - Z^{(k)}\right) \odot U + Z^{(k)} \odot V, \quad k = 2, \ldots, N_L \tag{4.6e}$$

$$y = W A^{(N_L)} + b \tag{4.6f}$$

This structure implements a neural network with N_L hidden layers, all containing the same number of neurons N_n. 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 X is projected into a higher dimensional space through the encoders layers, by using their respective weights and biases W^1, W^2, b^1 and b^2, resulting in the N_n dimensional vectors U and V. ϕ is an activation function, here the hyperbolic tangent function. The first step of calculating the forward pass is a normal pass through a dense layer to calculate $Z^{(1)}$, as shown in Equation (4.6b). Then this $Z^{(1)}$ is weighted by element wise multiplication ($\odot$) with the encoder vectors U and V in Equation (4.6c), to calculate the activation $A^{(1)}$ of the first layer. This propagation continues through all the remainder of the N_L layers. The final output y is calculated as a normal pass through a dense layer without activation function and use of the encoder layers.

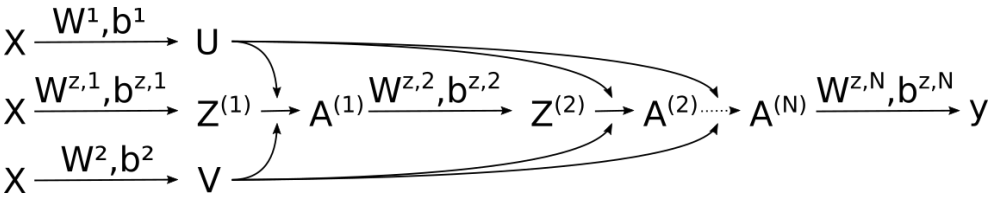


Figure 4.6: Illustration of the improved neural network structure for PINNs, as proposed by [6]. X is the input to the neural network, which is projected into a higher dimensional space through the two encoder layers, resulting in the higher dimensional vectors U and V. For each layer of the main network, we first calculate the intermediate activation Z utilizing the weight matrix and bias vector of this layer, then this Z is weighted by the encoder outputs U and V to calculate the final activation A of the layer.

4.4.1 Comparing the gradients of the improved and the normal structure

A new neural network with this new structure was trained to the same error on the validation set as the neural network analyzed in Figure 4.4. Figure 4.7 shows a comparison of the distribution of the gradients from the first term of the residual function $\frac{\partial \mathbf{y}}{\partial t}$ between the traditional and the improved structure. That is, the term $\frac{\partial \mathbf{y}}{\partial t}$ is differentiated with respect to all the weights and biases. In the first layer, the gradients are better for the traditional model, but for all other layers, the improvement is huge. Remember that a spike in the curve indicates that many of the gradients are close to zero, so the flatter curve is preferable.

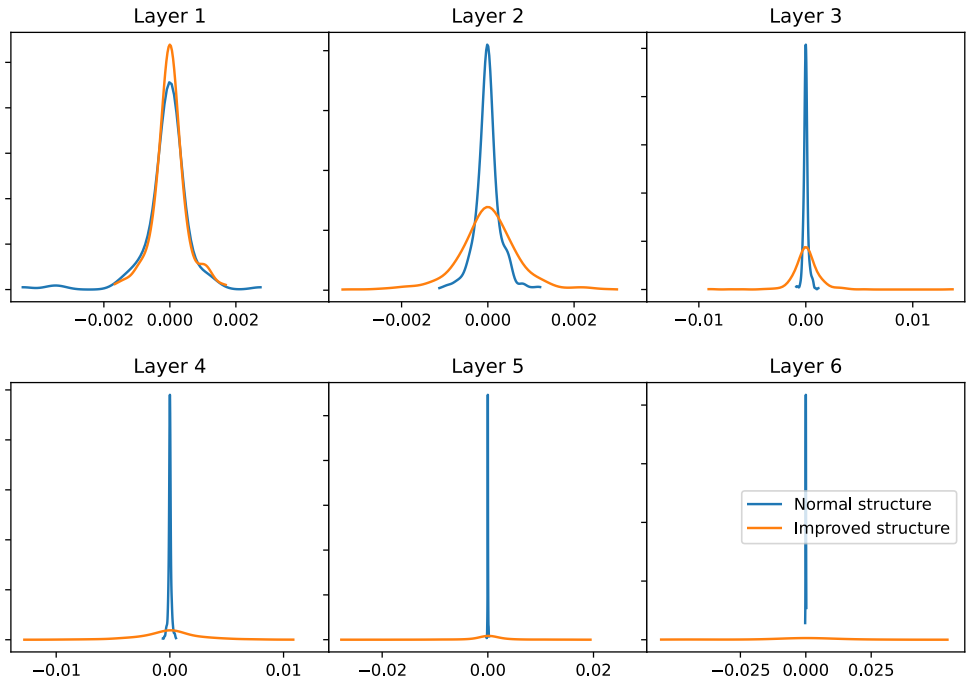


Figure 4.7: KDE plot comparing the distribution of the gradients with respect to the weights and biases through the hidden layers of two neural networks for the first term of the residual function: $\frac{\partial \mathbf{y}}{\partial t}$. Both neural networks are trained to the same error on the validation set, one using the basic dense network structure (blue) and one using the improved structure (orange). Except for in the first layer, the orange line is much flatter, indicating better gradients for the improved neural network structure. For example, in layer 6, the average magnitude of the gradients for the normal structure is $6.92 \cdot 10^{-5}$, while for the improved structure it is $9.53 \cdot 10^{-3}$. I practice, this offers a huge improvement on how much we progress at every training iteration, resulting in faster training of the neural network.

The same type of plot for the second term, $\mathbf{f}(\mathbf{y}, \mathbf{u})$, of the residual function is shown in Figure 4.8, which represents the gradients through the ODE. Here, the new

structure offers huge improvements in all the layers.

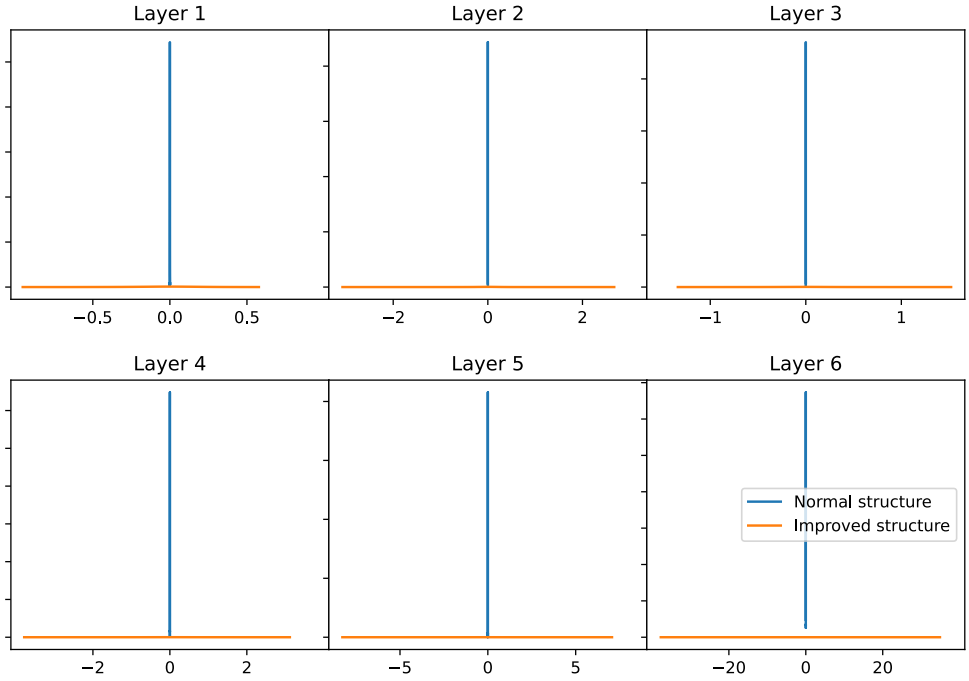


Figure 4.8: KDE plot comparing the gradients with respect to the weights and biases for the second term of the residual function, $\mathbf{f}(\mathbf{y}, \mathbf{u})$. The difference between the two types of neural network structure is so big that the distributions of gradients for the normal structure (blue) seems like vertical lines, compared to the orange horizontal lines for the improved structure. This indicates that there is a huge difference in magnitude of the gradients between the normal and improved structure. For the final layer, 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.

Comparing the values on the x-axis of the last two plots (4.7 and 4.8), it seems like the second term of the residual function is most important also for the improved neural network structure. Figure 4.9 verifies this by plotting the KDE of both terms of the residual function along with the KDE of the whole residual function of the neural network with the improved structure. From this plot it is clear that also for this improved structure the gradients of the second term of the residual function is the more important one, as for the traditional fully connected neural network. The difference is even greater for this new structure, the gradients of the first term seem to be negligible.

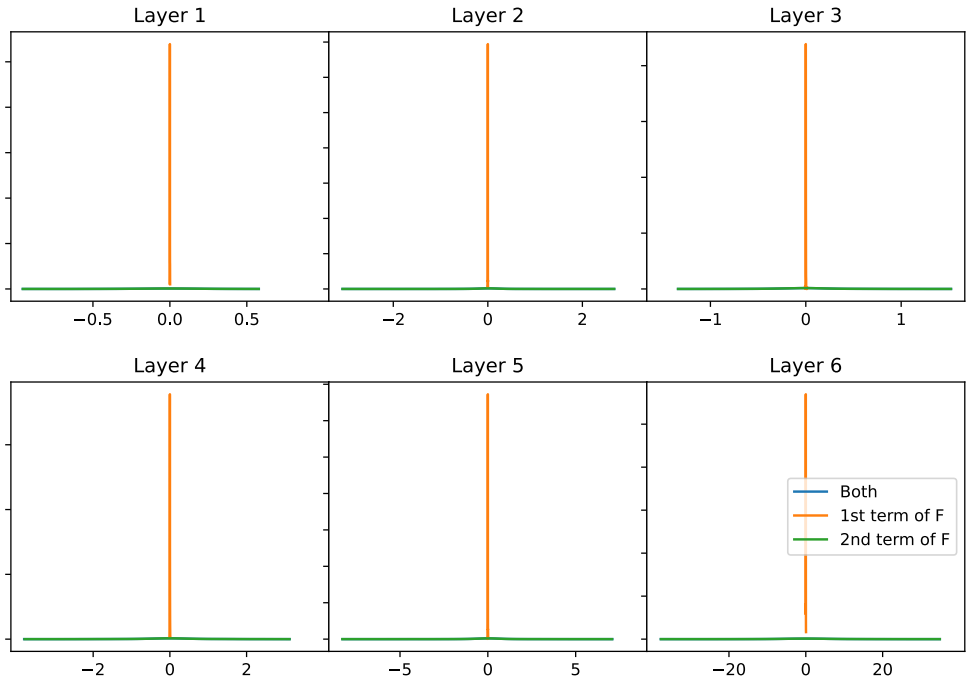


Figure 4.9: KDE of the gradients of the residual function with respect to the weights and biases of a neural network utilizing the improved structure. The blue line showing the KDE of the whole residual function is completely superimposed by the green lines, which shows the KDE for the second part of the residual function. This shows that the first term (in orange) contributes very little to the total magnitudes of the gradients.

This section has showed that the improved neural network offers huge improvement on the magnitude of the gradients with respect to the weights and biases, which is very important for training performance. We have also seen that it is extremely important to utilize the ODE term of the residual function when calculating the gradients, so modifying the well ODE as described earlier in this chapter was well worth the effort.

4.4.2 Comparing training of the improved and the normal structure

More interesting in practices, however, is how this new structure performs during training of the neural network. Figure 4.10 shows the validation loss during training of 10 networks of the traditional densely connected type in blue, and 10 networks with the new improved structure in gray, the dots indicate the endpoint of the training.

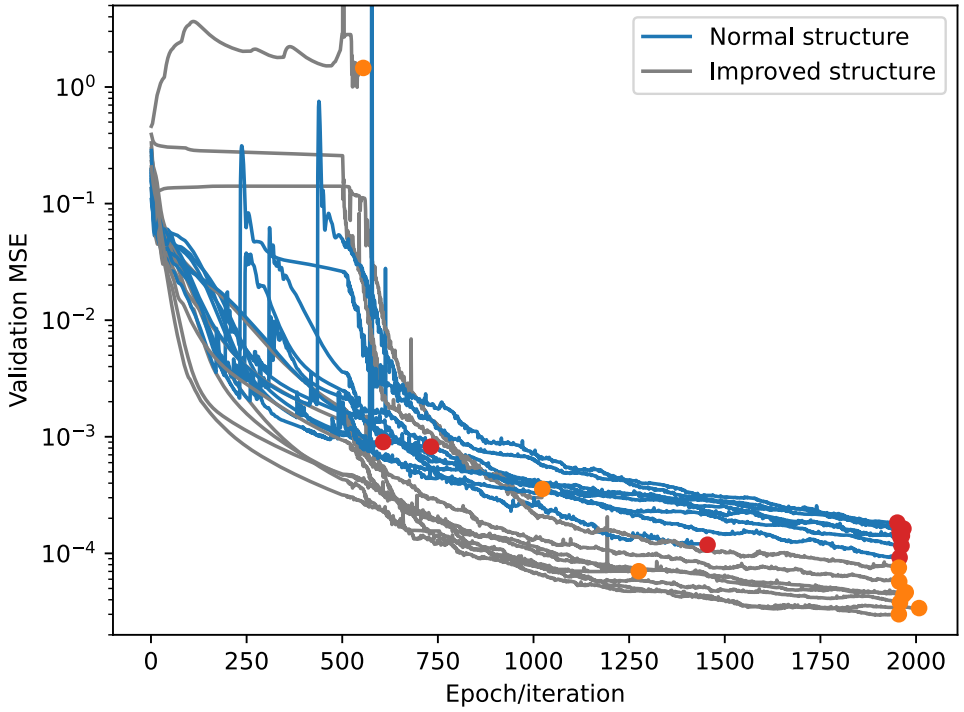


Figure 4.10: Validation MSE through training of 20 neural networks, 10 with the normal dense structure, and 10 with the improved structure. All networks were trained for 500 epochs of Adam, followed by 500 iterations of L-BFGS (these 500 iterations are shown as almost 1,500 in the plot due to the usage of the TensorFlow Probability implementation of L-BFGS, see main text for further explanation). The 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.

All networks were trained first with 500 epochs of Adam, then 500 iterations of L-BFGS. Six neural networks, three of each type, converged before completing the 500 iterations of L-BFGS, a common problem with this model. Here, the L-BFGS implementation from the TensorFlow Probability library is used. It does not have a feature for a callback function after every iteration, so logging here is performed at every call to the evaluation function (which computes the cost and gradients), performed on average around three times per iteration, explaining why the 500 iterations of L-BFGS extend from 500 to almost 2000 in the plot.

The interesting part of Figure 4.10 is that for all the seven networks where the training does not stop early, the new structure outperformed the normal structure. A clear indication that this is indeed an improved structure, at least for this model. However, formal experiments training the neural networks until convergence was not conducted due to prioritizing other experiments. This would be interesting, as the final performance is the most important measure.

4.5 Computing the algebraic variables

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

The first idea is to simply use the equations from the model to calculate the algebraic variables from the states. Initially this seemed like the best approach, as it does not add complexity to the PINC, and the algebraic variables will match exactly the state prediction. However, in practice, this approach was not effective as the optimization algorithm in CasADi usually crashed before completing a meaningful simulation, and there was no specific pattern to the crashes. Most likely, the nature of the equations, with square roots and maximum operators, led the optimizer to crash from time to time.

The second approach, made to avoid these crashes, was to train another neural network to predict the algebraic variables from the states. The input to this neural network is the states and the control input, the output is one or more of the algebraic variables of the model which are of interest. This setup is illustrated in Figure 4.11.

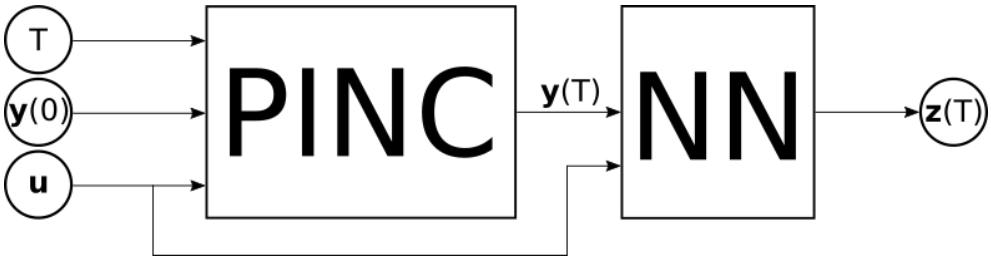


Figure 4.11: Demonstrating how an additional neural network was applied to predict algebraic variables of the oil well, for instance the bottom-hole pressure. The input to this additional network is the state predictions from the PINC and the control input.

The prediction of these algebraic variables could also be implemented directly into the PINC, but with this approach we can train the new neural network independently of the PINC. If we want to utilize different algebraic variables in the MPC, we can simply train a new neural network for this, without having to retrain the PINC model, which is a much slower process.

The neural network will be trained to predict four algebraic variables:

- *bottom-hole pressure* – which is the subject of control for one MPC application.
- *Gas-lift injection rate* – which can be a limited resource the controller needs to account for.

- *Gas production* – which may need to be limited as some plants have limited capacity for processing gas.
- *Liquid production* – which can be used for economic optimization.

4.5.1 Training and hyperparameter search

This new neural network is trained independently of the PINC. As there is no dynamics involved in calculating the algebraic variables from the states, there is no need for the PINN framework. Training data is sampled from the entire state- and control input space, in the same manner as for the PINC. The corresponding target outputs (the values of the algebraic variables) are calculated using the ODE equations.

A hyperparameter search was conducted to find a structure that gives good predictions. The structures tested were fully/densely connected neural networks with 1-4 hidden layers with 10, 20, 30 and 40 neurons per layer. Ten tests were conducted for each configuration, with different weight initialization, and trained for 5,000 iterations of L-BFGS (which seemed to train faster than Adam). The selected structure was 4 hidden layers of 30 neurons each.

A neural network of this structure was then trained until convergence with L-BFGS, resulting in almost perfect predictions of these four algebraic variables, as will be shown in the next chapter. Only the prediction performance was taken into account when selecting the neural network structure, not the size of the network. It may have been sufficient to select a smaller network, which would result in faster computations of the MPC, but this was not investigated further.

4.6 MPC implementation in CasADi

The MPC controller will be implemented using CasADi, an open-source numerical optimization framework. The official CasADi documentation [15] offers an introduction to the framework, including how to formulate and solve optimal control problems. Only the most important parts of the code will be displayed here.

The prediction horizon of the MPC (N) will vary depending on the control problem, while the time step T of the predictions will be $T = 60$ seconds for all the control applications. The prediction horizon is chosen so that the system reaches a steady state within the prediction horizon. For the state tracking, the system settles quickly, so a prediction horizon of 10 minutes ($N = 10$) is sufficient. For controlling the bottom-hole pressure, however, it takes a lot longer for the states to reach steady state, even though the pressure reaches its reference quickly, here the prediction horizon will be 50 minutes.

When controlling the bottom-hole pressure, this pressure may reach its reference without the states stabilizing, because there are infinitely many combinations of state values and control inputs that will keep the bottom-hole pressure at the

reference. If the states run too far off, we may end up in an area of the state space where the model is not defined, or the MPC is not able to find a solution to the optimization problem. To ensure that the end of the prediction reaches a steady state, the control input horizon is selected shorter than the prediction horizon ($N_u < N$). In practice, this means that the control input will be constant for some of the last steps of the prediction. If the system is not at a steady state by then, the bottom-hole pressure will drift off from its reference. Also, through the weight matrix **Q**, an extra penalty is added to these final reference deviations, making the MPC prioritize finding a stable state more than the transient response. A much longer prediction horizon is needed here compared with the state references, as the states takes a long time to settle. Setting the prediction horizon to $N = 50$ and control input horizon to $N = 45$, in addition to increasing the **Q** matrix weight for the final 5 steps, yielded good control performance.

The MPC will be formulated as a general Non-Linear Programming (NLP) problem using the `casadi.Opti` class. In the following code listing, an instance of this class is created in line 1, then variables and parameters are added to the `Opti` object:

Code listing 4.1: Implementing a NN as a CasADi function.

```
1  opti = casadi.Opti()
2  N_state=3
3  N_input=2
4  x = opti.variable(N_state, N+1)
5  u = opti.variable(N_input, N)
6  du = opti.variable(N_input, N_u)      #change in u for each step
7  P_bh = opti.variable(1, N)
8
9  x0 = opti.parameter(N_state,1)
10 u_last = opti.parameter(N_input, 1)
11 P_bh_ref = opti.parameter(1, N)
12 Q = opti.parameter((N+1), (N+1))
13 R = opti.parameter(N_u*N_input, N_u*N_input)
```

Here, `N_state` is the number of states and `N_input` the number of control inputs (production choke and gas-lift valve). This setup is for controlling the bottom-hole pressure. The `opti.variable()` type are variables of the optimization problem, while `opti.parameter()` are parameters that are given to the MPC at every iterating, 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 detail of the implementation is how to ensure that the variables `x` and `P_bh` satisfy the system dynamics. Code listing 4.2 shows that the function `opti.subject_to()` is used only on the state variable, this constrains the solution to satisfy the system dynamics. Here, `F()` is a function that maps a state and control input to the state 60 seconds ahead in time, the PINC will be used to provide this prediction. For the bottom-hole pressure, there is no need for a constraint, an assignment operator is enough. If we use the `opti.subject_to()` here as well,

we add unneeded complexity to the optimization problem, which results in slower computation.

Code listing 4.2: Ensuring that the MPC solution satisfy the system dynamics.

```
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]))
4
5 for i in range(0, N):
6     P_bh[:,i] = F_x_to_P_bh(x[:,i+1], u[:,i])
```

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:

Code listing 4.3: Implementing of control input change.

```
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])
```

The first line ensures that the last applied control input is taken into account, 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 iteration of a simulation, we either 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 can be useful when starting a simulation from an initial condition where a steady state control input is not known.

Chapter 5

Results

This chapter will display the results obtained in this work, including: the training progress of the PINC, long range simulations with the PINC, prediction of bottom-hole pressure through the additional neural network, and MPC applications controlling the states and the bottom-hole pressure utilizing both NMPC with the PINC and SLMPC. The results will be discussed further in the next chapter.

5.1 PINC model of the oil well

5.1.1 Training

The training loss and validation loss during training of the PINC for the first well is shown in Figure 5.1. The final training loss is $7.48 \cdot 10^{-7}$, and the validation loss is $1.43 \cdot 10^{-6}$. These values are however not directly comparable to each other, as the nature of how they are calculated are different. The training loss is composed of the loss of the initial condition and physics based loss, while the validation loss shows the average (scaled) error for 100 predictions 60 seconds ahead in time. The PINC was trained with an initial 1,000 epochs of Adam, then until convergence with L-BFGS. The structure of the PINC is 6 hidden layers of 40 neurons each, as found in the hyperparameter search.

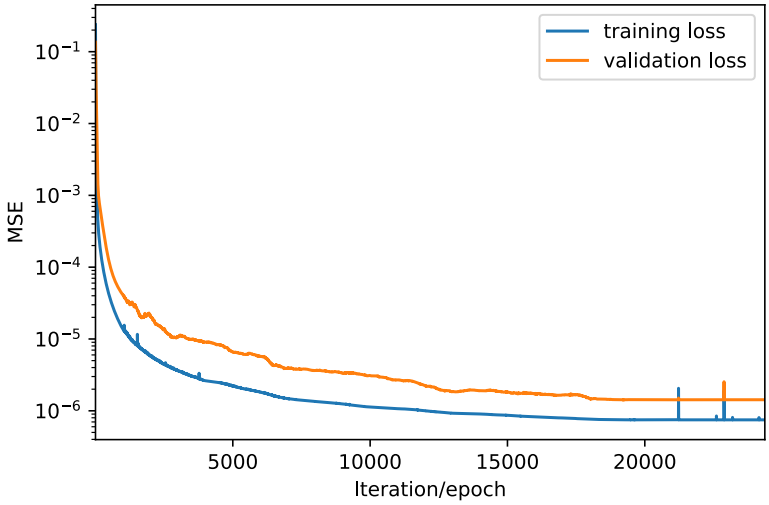


Figure 5.1: Plot of the training and validation loss of the PINC trained to predict on well 1. The PINC was trained for 1,000 epochs of Adam, then until convergence with L-BFGS. As the nature of how the training and validations loss are computed are different, the numeric values of these two losses are not directly comparable. The final training loss is $7.48 \cdot 10^{-7}$, and the validation loss is $1.43 \cdot 10^{-6}$.

The training of the PINCs for the other wells will not be displayed, they showed similar training performance to the first well. The same neural network structure of 6 hidden layers of 40 neurons was used for all the wells, without conducting separate hyperparameter searches for each well.

5.1.2 Prediction performance in self-loop

The performance of the PINCs was evaluated by doing 100 test simulations of 3,000 seconds each, with a sequence of random control inputs that change every 60 seconds, using the IAE as a performance measure. The PINC was put in self-loop for 50 iterations to obtain these simulations. The network was only fed the initial condition and the sequence of control inputs, so error in predictions will accumulate over the self-looping. Figure 5.2 shows one of the test simulations for the first well. In blue is the Runge-Kutta simulation which represents the real system's behavior, in dashed pink line is the prediction of the PINC, the larger pink dots indicate the output of the PINC after each iteration of self-loop. The bottom most plot shows the randomly generated control input sequence.

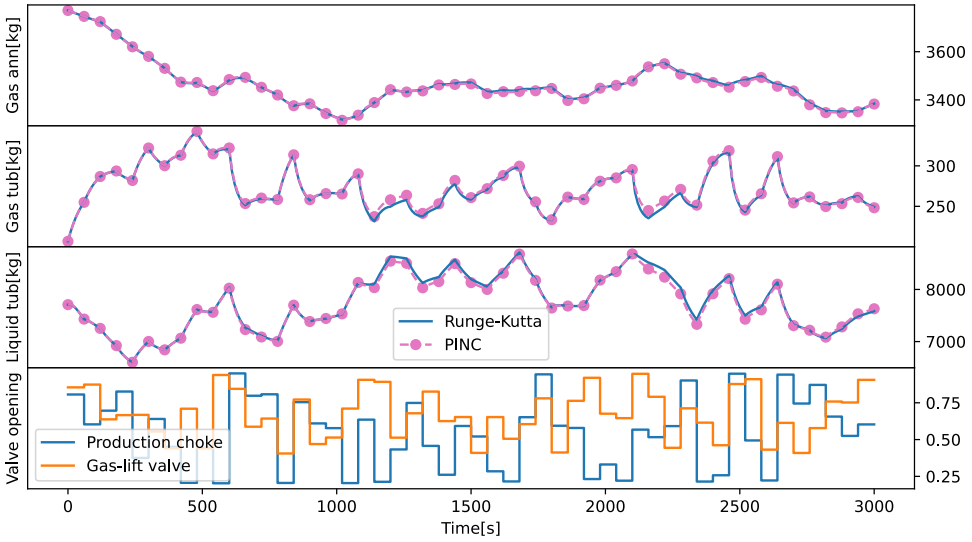


Figure 5.2: A test simulation of 3,000 seconds for the first well. A random sequence of control input, changing every 60 seconds, and a random initial condition are passed to the PINC. The resulting predictions are shown in pink, with the dots indicating each self-loop iteration. The blue line (in the upper three plots) shows the real value of the states, obtained by Runge-Kutta simulation. The bottom plot shows the randomly generated control input sequence. The IAE of this test simulation is 0.00959, a bit below the average, as shown in Table 5.1.

Table 5.1 shows the IAE of these 100 test simulations, in average and standard deviation (of scaled variables). The average IAE for the first well is 0.00994, for comparison the IAE of Figure 5.2 is 0.00959, and the highest obtained IAE of these 100 test sets for the first well was 0.0332, which is shown in Figure 5.3.

IAE	Well 1	Well 2	Well 3
average	0.00994	0.00448	0.00185
standard deviation	0.00621	0.00345	0.000325

Table 5.1: Average and standard deviation of the IAE for 100 test simulations of 3,000 seconds.

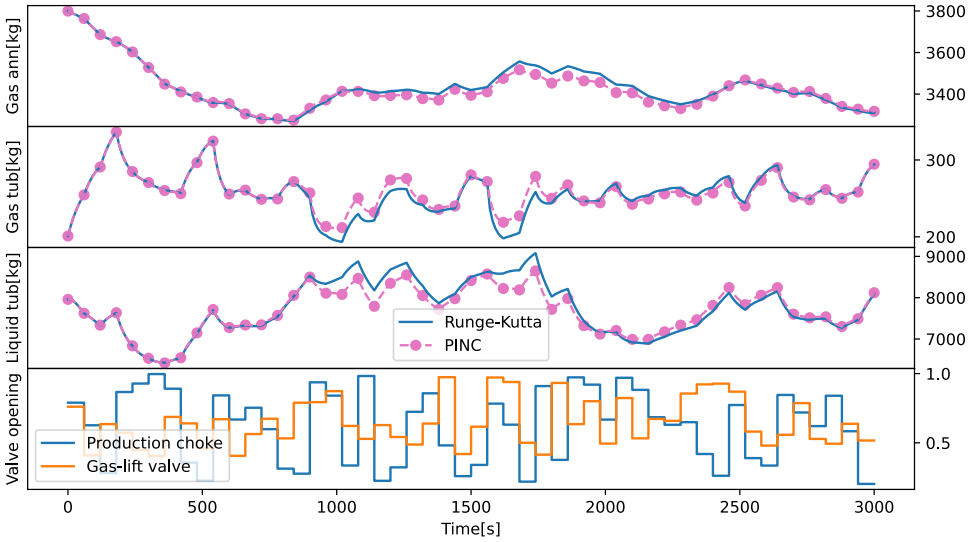


Figure 5.3: The test simulation obtaining the highest IAE of the 100 test sets for the first well. The IAE is 0.0332.

5.1.3 Predicting the bottom-hole pressure

The extra neural network for predicting the bottom-hole pressure from the states and control input was trained until convergence with L-BFGS. To evaluate the performance of this neural network, a validation set was constructed by randomly sampling $10,000$ points from the input space, the neural networks predictions for these points were compared with their true values (computed using the ODE equations). For the first well, the IAE on this validation set was 0.00763 bar. This neural network was also applied to the previously shown test simulation of 50 iterations of PINC self looping. Figure 5.4 shows the same test simulation as Figure 5.2, with the addition of the bottom-hole pressure estimates. Here, the error of the bottom-hole pressure predictions will contain both the error accumulated through the PINC self looping, and the error of this extra neural network. For the 50 test sets, the average IAE for the first well is 0.3087 bar. Table 5.2 shows the IAE on the validation set and test sets for all the wells.

	Well 1	Well 2	Well 3
Validation IAE [bar]	0.00763	0.00744	0.0109
Test set average IAE (w/PINC)	0.3087	0.2497	0.0656

Table 5.2: IAE for all wells on the validation set and the 50 self loop test sets. The validation set contains randomly sampled state and control input values, the predictions of this input are compared with the target pressure which is calculated through the ODE equations, so this validation error represents the error of the bottom-hole pressure predicting neural network. The IAE on the test set is caused mostly by the accumulation of error in the self looping of the PINC.

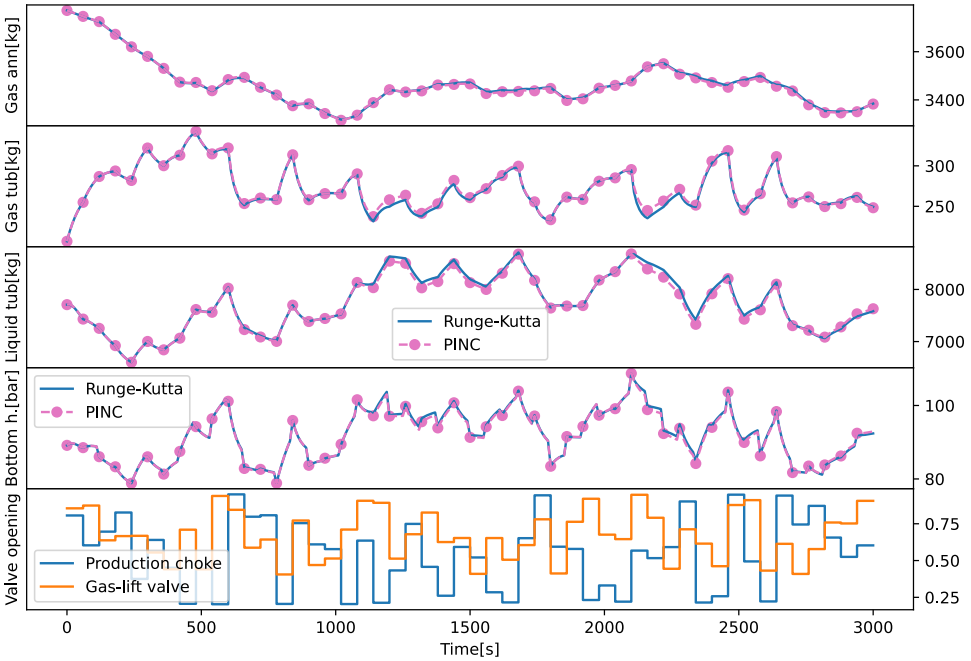


Figure 5.4: Displaying the same test simulation as shown in figure 5.2, now with the addition of the bottom-hole pressure predictions, which are generated by passing the PINCs predictions through the additional neural network made to predict the algebraic states. So the errors in pressure predictions is a combination of the accumulated error of the PINC in self-loop, and the error of this additional neural network. The IAE of the pressure is 0.2813 bar.

5.2 MPC of single well with state reference

In this experiment, two of the states are controlled, the mass of gas and liquid in the tubing. The MPC has a prediction horizon of $N = 10$, and the control input is allowed to change for the entire horizon, $N_u = 10$. The following weight matrices were used:

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} 10^7 & 0 \\ 0 & 10^6 \end{bmatrix} \tag{5.1}$$

The first element of $\mathbf{Q}$ is 0 to indicate that the first state is not controlled. Figure 5.5 shows the resulting response of the PINC MPC and the SLMPC for two step changes in the reference. The IAE for the PINC MPC is 11.90 and for the SLMPC is 7.47.

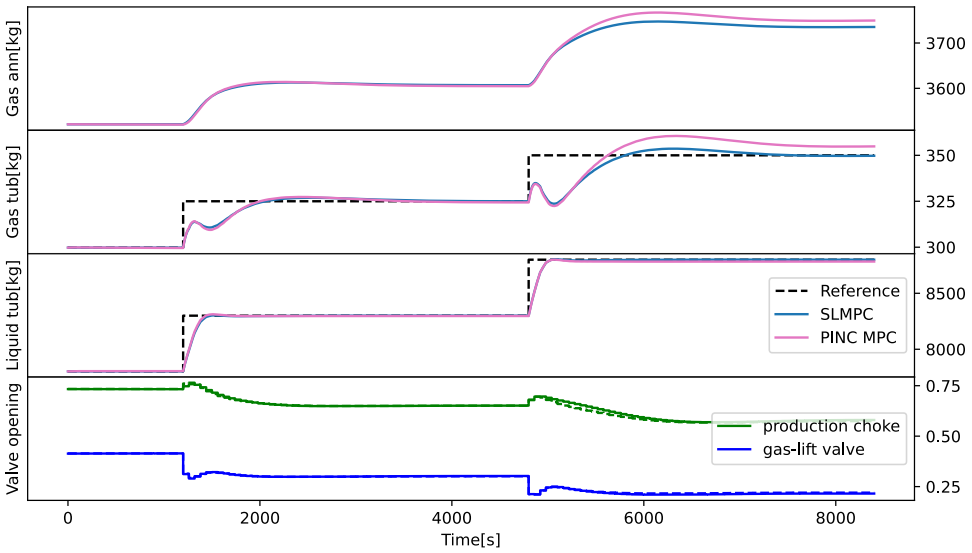


Figure 5.5: Controlling the mass of gas and liquid in the tubing, with two step changes in reference. The system states for the PINC MPC is shown in pink and for the linearized MPC in blue. For the control input in the bottom most plot, the PINC MPC control input is shown in solid lines, and the SLMPC in dashed lines. IAE PINC: 11.90, IAE SLMPC: 7.47.

It can be seen from Figure 5.5 that the PINC MPC settles with some steady state deviation for both controlled states. This is due to error in the predictions of the PINC. Figure 5.6 demonstrates this by showing the last 600 seconds of the simulations along with the future predictions of the MPC at the end point of the simulation. The pink dotted line shows that the PINC MPC predicts that the states will approach the reference already at the next iteration, but from the past values we can see that the states already have settled. We can also see that the SLMPC also has some error in the predictions, although to a smaller extent.

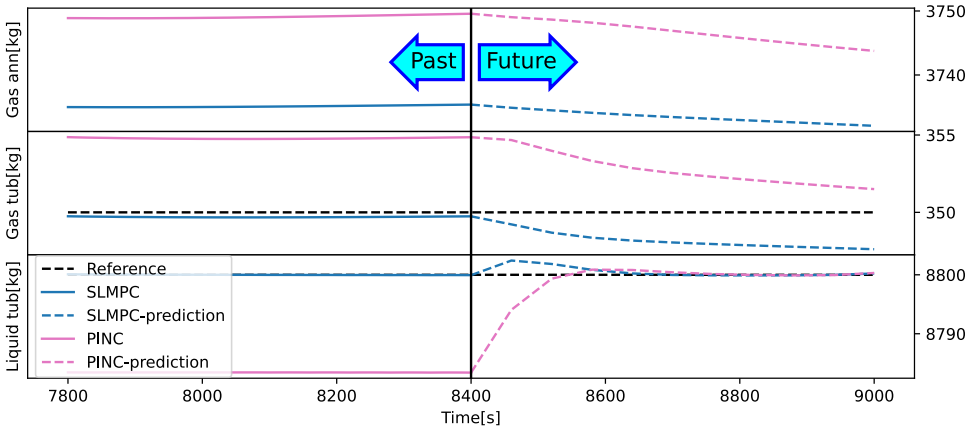


Figure 5.6: A closer look at the end of the simulation of Figure 5.5, along with the predictions of the MPC at the end point. The vertical black line shows the current time, the solid lines to the left are the system states of the past for both controllers, and the dashed lines to the right are the future predictions of the controllers. For the PINC MPC we can see that the prediction is that the liquid mass in the tubing will reach its reference in a few steps, and that the gas in the tubing will also approach its reference, but the system is already at steady state, so the deviation seen to the left (the past) is due to imperfections in the PINCs predictions. Note that the scales are in kilos, so the deviations are quite small.

5.3 MPC of single well with bottom-hole pressure reference

When controlling the bottom-hole pressure, the states take a longer time to settle, so the prediction horizon was increased to $N = 50$. To ensure that the end point for the prediction is stable, the input is not allowed to change for the last five iterations ($N_u = 45$), the five last bottom-hole pressure predictions are also penalized extra through the weight matrix $\mathbf{Q}$:

$$Q_{ii} = \begin{cases} 1 & i \in [1, \ldots, 45] \\ 100 & i \in [46, \ldots, 50] \end{cases}, \quad R = \begin{bmatrix} 10^3 & 0 \\ 0 & 10^3 \end{bmatrix} \tag{5.2}$$

Figure 5.7 shows a simulation with two step changes in bottom-hole pressure reference. The states are shown in the figure to demonstrate that they settle, they are not controlled or constrained in any way. The initial condition of the system also deviates from the settling point of the first reference. This initial response is maybe the biggest challenge for the controllers in this experiment. The IAE of the PINC MPC is 0.038 bar and for the SLMPC it is 0.058 bar.

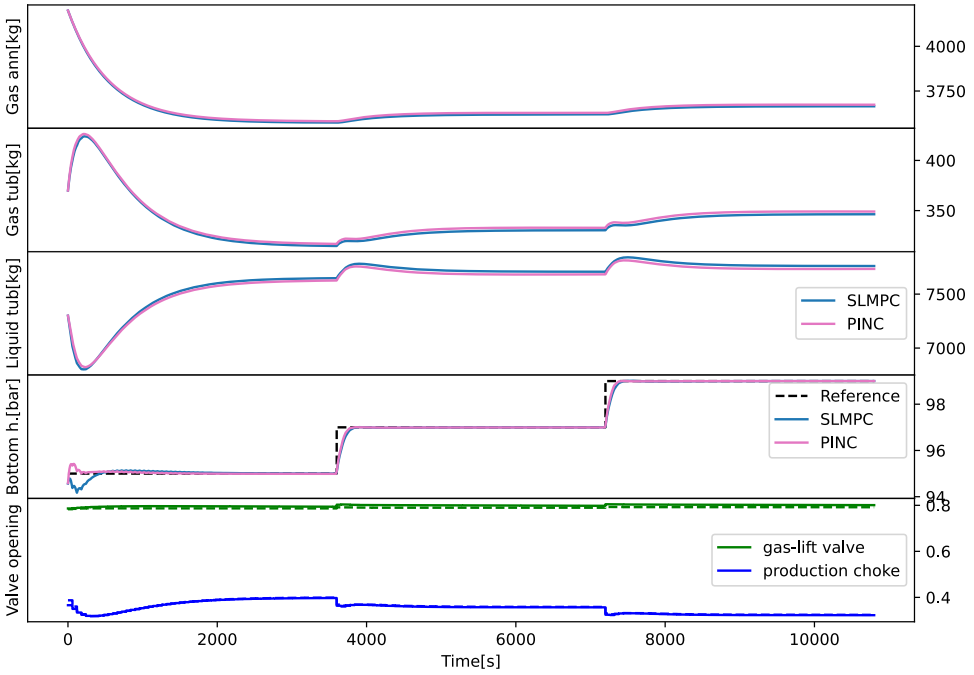


Figure 5.7: Simulation of three step changes of 2 bar to the reference of the bottom-hole pressure (95 to 99 bar), for the PINC MPC in pink and the SLMPC in blue. For the control input in the bottom most plot, the PINC MPC control input is shown in solid lines, and the SLMPC in dashed lines. The upper three plots show the states, which are not controlled or constrained. The IAE of the PINC is 0.038 [bar], and of the SLMPC it is 0.058 [bar].

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. An example of these simulations is shown in Figure 5.8.

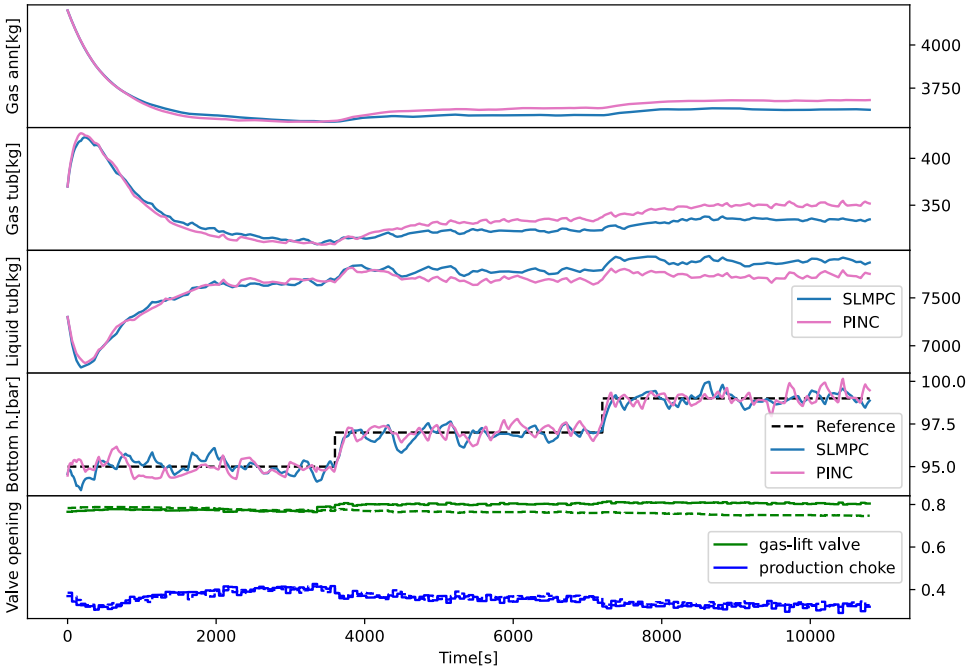


Figure 5.8: Same simulation as the previous plot, now with the addition of measurement noise. In the bottom plot, the PINC MPC control input is shown in solid lines, and the linearized MPC in dashed lines. The IAE of the PINC is 0.317 [bar], and of the SLMPC it is 0.327 [bar].

Several of these simulations were completed, and the IAE was computed for each. The resulting average and standard deviation statistics are shown in Table 5.3, along with the IAE of the noiseless experiment.

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

Table 5.3: IAE of simulations with the two MPCs. The 0% noise experiment is the one shown in Figure 5.7. While the 5% noise statistics are calculated from several simulations, one of which are shown in Figure 5.8.

From the simulations with measurement noise, the average response and the standard deviation was calculated along the trajectory, which is shown in Figure 5.9. The figure shows in solid line the average response, and the two shaded areas show the one and two standard deviations regions, which are respectively where approximately 68% and 95% of the simulations will be.

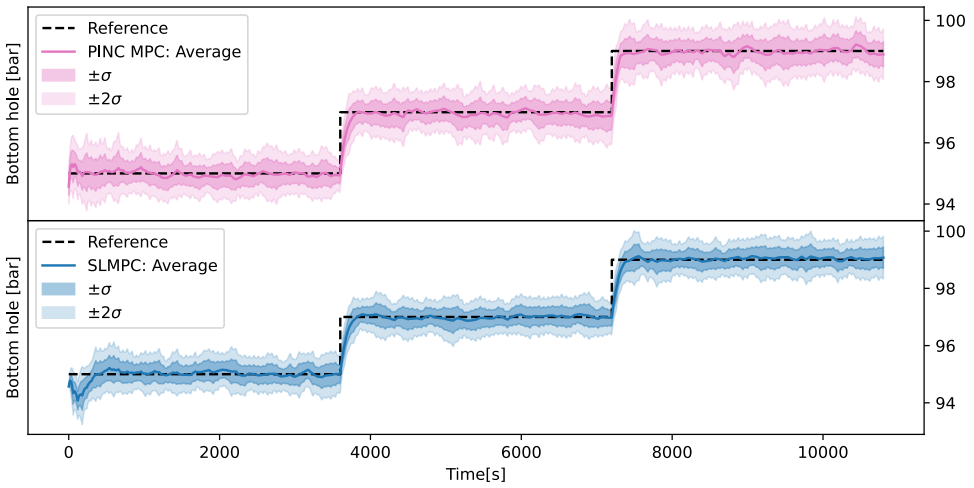


Figure 5.9: After running several simulations with measurement noise, the average response along with the standard deviation was calculated for booth controllers. The upper plot shows the PINC MPC and the lower plots shows the SLMPC. The innermost shaded area shows the standard deviation of the response, approximately 68% of simulations will be within this interval. The outer shaded area shows the two standard deviation of the response, approximately 95% of simulations will be within this area. Along the trajectories, the average standard deviation for the PINC MPC is 0.392 and for the SLMPC it is 0.333.

The computation times needed to compute one step of the MPC algorithm is approximately 10 seconds for the PINC MPC, and less than one second for the SLMPC, computed on an old laptop. Both well within the $T = 60$ seconds between each MPC calculation.

Chapter 6

Discussion

In this chapter, the results and some implementation details shown in the two previous chapters will be discussed. Starting with the training of the PINC model, then discussing the resulting prediction performance of the PINC, and finally the MPC applications.

6.1 Training of the PINC

By far the biggest challenge in this project was to achieve reliable successful training of the PINC on the oil well model. Initially, even training a traditional PINN with fixed initial condition and control input seemed very difficult. Usually the training of the neural networks would converge within a few minutes, and the obtained performance of these neural networks were nowhere near useful. A lot of trail and error were applied to obtain a PINC model with a useful prediction accuracy. As so many things were tried during the project, it is hard to evaluate how much each individual component contributes, but it is certain that the combination of the methods that were applied offers a huge improvement. The training of the PINC is now reliable for the oil well model, the training failing only approximately 20% of the time. And if a neural network fails to train properly, it fails early in the training process. Training a handful (5-10) of neural networks for a short period, then selecting the best one and train this until convergence, seems to offer good results.

As for the changes made to improve the training of the PINC, the addition of the maximum operator within the square roots of the ODE was very important, as the neural network would fail to train without it due to the gradients not being defined. And the lower limit of this maximum operator applied within the square root function of 10^{-3} turned out to be very important, for instance using 10^{-5} offered much poorer results. This is a method that can be very useful for any ODE that involves functions that are not defined for negative numbers (or some other

limited domain). In addition to using the maximum operator, an attempt was also made to extend the square root function for negative numbers by making an odd extension, with the goal to maintain gradients also for negative inputs, but this offered no improvement over simply truncating negative numbers with the maximum operator. However, for other problems, it may be worthwhile to investigate different methods of coping with undefined domains of functions.

The approximation of the friction term is another useful tool that can be of great use for other systems. Not only other systems that include this friction term, but in general, any complicated term can be replaced by an approximation that inhibits better gradient properties.

When the PINC for the second well was trained, it was discovered that it was hard to construct limits for the states that included all the regions of the state space encountered in normal operation without including infeasible regions. The method of removing the training data points that were infeasible offered a simple and elegant solution, without spending time on constructing a more complex domain for the state space. This is another universal tool that can be useful for other applications. For the second oil well model, this was crucial for successful training. The downside of this method is that we can still draw collocation points where the initial condition is feasible, but the system will move into infeasibility during the collocation points time, so that the target state is not defined. This method does not eliminate these kinds of points, which are present to an unknown extent. As the resulting performance of the PINC was satisfactory, this was not further investigated.

In the search for overall better performance during training, the improved neural network structure with skip connections was found. It was developed specifically for PINNs, and offered a big improvement for this model. No other network structures were tested, so this may be an interesting topic for future research, with the potential of offering massive improvements.

6.2 Prediction performance

The test simulations containing a sequence of 50 randomly drawn control inputs show that the PINC is able to reproduce the system's behavior quite well on average, and even reasonably well for the worst example (shown in Figure 5.3). Each of these test simulations last for 50 minutes, which represents the prediction horizon for the MPC controlling the bottom-hole pressure, so a good prediction accuracy for this time window is desirable. These randomly drawn input sequences have a big variety and fast change of control inputs, which may not be very realistic in a real application. The reasoning is that if the PINC is able to predict well on this kind of test set, it should also be able to predict well for any kind of smoother control input sequence, for instance as computed by the MPC. The two other wells that were constructed by adjusting the parameters of the model achieved even bet-

ter performance on these test sets (Table 5.1). No control applications were done for these two additional wells due to lack of time.

As the systems equations offered problems when used in the MPC formulation to calculate the bottom-hole pressure (CasADi would crash during optimization), the additional neural network that was trained to predict the bottom-hole pressure offered a nice solution. The neural network is here used as a sort of universal approximation to a known function. This may be useful in other applications, effectively removing all the system equations and replacing them with a neural network, for which we know we can calculate the gradients relatively easy and efficiently. The resulting performance of these neural networks for predicting the bottom-hole pressure is very good, with less than 0.01 bar average error in predictions, which is much less than the error of the PINCs predictions. We could probably have obtained satisfactory results by using a smaller neural network structure for this additional network. The structure used here consists of 4 hidden layers of 30 neurons each. The neural network was also trained to predict four of the algebraic states, but only the bottom-hole pressure was used for the experiments shown here, the three other states were intended for a control scenario where one MPC would control several wells to optimize oil production with limits on the available gas lift. If the neural network was only trained to predict the bottom-hole pressure, instead of all four, that would probably also reduce the needed size of the neural network.

If the PINC were to be used in a real control application, it might be better to include the prediction of the bottom-hole pressure (and other variables of interest) in the PINC network to avoid added complexity and computing time during MPC calculations. In general, during the hyperparameter search for the PINC, it might be interesting to take into account the computational time needed for different neural network structures to compute the MPC algorithm. If a simpler neural network structure can obtain satisfactory prediction accuracy, and it also decreases the computational need of the MPC algorithm, this simple structure would be a better choice. The hyperparameter searches here only analyzed the initial training response, so it is possible that a simpler neural network structure could obtain the same (or better) prediction accuracy, but that training is just slower. In this case, we could choose to spend more time on training to save time and computational power during control. These are questions that may be very interesting when applying this method to systems where computational time of the controller is a challenge.

6.3 MPC experiments

First and foremost, the experiments show that MPC utilizing the PINC is feasible for the oil well model. The main aim of this work was to extend the PINC framework to this new model, and not to improve upon previously existing controllers. The SLMPC shows over all better control performance, and has a much faster

computation time. The only area where the PINC MPC performed better than the SLMPC were some preliminary experiments with very large change in reference, this may be due to the linearization not being accurate for large changes during the control horizon, while the PINC has good accuracy for the entire region of interest. But these experiments were not worked out thoroughly enough to put in this report, and these large changes in reference would probably not be likely in a real control application of an oil well. As the goal of this work was not to improve upon current available controllers, the SLMPC will not be discussed further in this section, focus will instead be put on the MPC utilizing the PINC.

6.3.1 State control

The state control experiments mostly demonstrate that the PINC is usable with MPC. For this experiment it was also tried to use a Runge-Kutta integration instead of the PINC, but this would fail very often, crashing CasADi, without completing any meaningful simulations. The reason for the Runge-Kutta integrator not work was not found, but it may be that the ODE equations offer a too big challenge for the optimization algorithm that solves the MPC problem. This crashing never happened for the MPC utilizing the PINC, indicating that the neural network formulation is better suited for optimization than the ODE equations for this model. The control experiment shows some steady state deviation from the state references. Further investigation showed that this was due to imperfections in the predictions of the PINC. As the PINC did not obtain perfect prediction accuracy, it is to be expected that there will be some deviation when applying it to control. Looking at the scales of the states, this deviation is very small, only a few kilos' deviation from the reference. All in all, these results show feasibility of the PINC and MPC combination, and good control performance.

6.3.2 bottom-hole pressure control

These experiments show good control performance both for the experiment without measurement noise and for the experiment with measurement noise. For the noiseless experiment, the bottom-hole pressure settles close to the reference quickly, with little deviation. With measurement noise, the controller still acts as fast for change of reference, but is obviously not keeping as close to the reference, due to the measurement noise. However, the average, taken over several simulations of this same experiment, shows that the average response is keeping very close to the reference.

The plots from these experiments show that the bottom-hole pressure changes very quickly and that the states take a lot longer time to settle. The fast settling of the bottom-hole pressure while the states are still in rapid change may be due to the simplification made during the derivation of the model, as described in Section 2.4. The authors argue that these simplifications will matter little for the steady states case, which is what they aimed to control [5]. These simplifications

were made to avoid having an implicit set of equations, that is, to keep the model equations explicit. However, when using the PINC with MPC, we do not utilize the model equations, they are only utilized during training. It would be interesting to re-derive the oil well model without these simplifications, resulting in a Differential Algebraic Equation (DAE), training another PINC for this, and comparing the PINC trained using the simplified ODE and the PINC trained using the DAE in a control application run using an industry standard oil well simulator.

Chapter 7

Conclusion and further work

This report has shown that Physics Informed Neural Network for Control (PINC) are able to reproduce the behavior of the gas-lifted oil well model good enough to be used with in Model Predictive Control (MPC). Further, it showed that, with the addition of a second network that predicts the bottom hole pressure from the state predictions of the PINC, the MPC is able to control this pressure with good performance. In the process of developing the solution to the control problem, several challenges appeared, the solutions to these challenges, which have been described in this report, are arguably the main contributions of the report, as they may be useful tools when applying PINCs to other systems.

PINCs were trained for three oil well models, where the second and third were constructed by adjusting the parameters from the first well. The goal was to control all three wells with one MPC, having a limit on the total available gas-lift, and doing a step change in this gas-lift availability. This is a possible direction to extend the work undertaken here, despite offering little new results in the domain of PINC research, it but could be a nice proof of concept. Further, the framework of PINC may be applied to other models to move this field of research forward.

Bibliography

[1] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations,' 2017. DOI: 10.48550/ARXIV.1711.10561. [Online]. Available: https://arxiv.org/abs/1711.10561.

[2] M. Raissi, P. Perdikaris and G. E. Karniadakis, 'Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations,' 2017. DOI: 10.48550/ARXIV.1711.10566. [Online]. Available: https://arxiv.org/abs/1711.10566.

[3] E. A. Antonelo, E. Camponogara, L. O. Seman, E. R. de Souza, J. P. Jordanou and J. F. Hubner, 'Physics-informed neural nets for control of dynamical systems,' 2021. DOI: 10.48550/ARXIV.2104.02556. [Online]. Available: https://arxiv.org/abs/2104.02556.

[4] J. Nicodemus, J. Kneifl, J. Fehr and B. Unger, 'Physics-informed neural networks-based model predictive control for multi-link manipulators,' 2021. DOI: 10.48550/ARXIV.2109.10793. [Online]. Available: https://arxiv.org/abs/2109.10793.

[5] E. Jahanshahi, S. Skogestad and H. Hansen, 'Control structure design for stabilizing unstable gas-lift oil wells,' *IFAC Proceedings Volumes*, vol. 45, no. 15, pp. 93–100, 2012, 8th IFAC Symposium on Advanced Control of Chemical Processes, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20120710-4-SG-2026.00110. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016304256.

[6] S. Wang, Y. Teng and P. Perdikaris, 'Understanding and mitigating gradient pathologies in physics-informed neural networks,' 2020. DOI: 10.48550/ARXIV.2001.04536. [Online]. Available: https://arxiv.org/abs/2001.04536.

[7] J. E. Kittelsen, 'Physics-informed neural networks: A basic introduction and how it can be used with model predictive control,' Norges Teknisk-Naturvitenskapelige Universitet, 2021.

[8] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[9] Z. Mao, A. D. Jagtap and G. E. Karniadakis, 'Physics-informed neural networks for high-speed flows,' *Computer Methods in Applied Mechanics and Engineering*, vol. 360, p. 112 789, 2020, ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2019.112789. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045782519306814.

[10] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado and E. Kuhl, 'Physics-informed neural networks for cardiac activation mapping,' *Frontiers in Physics*, vol. 8, 2020, ISSN: 2296-424X. DOI: 10.3389/fphy.2020.00042. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fphy.2020.00042.

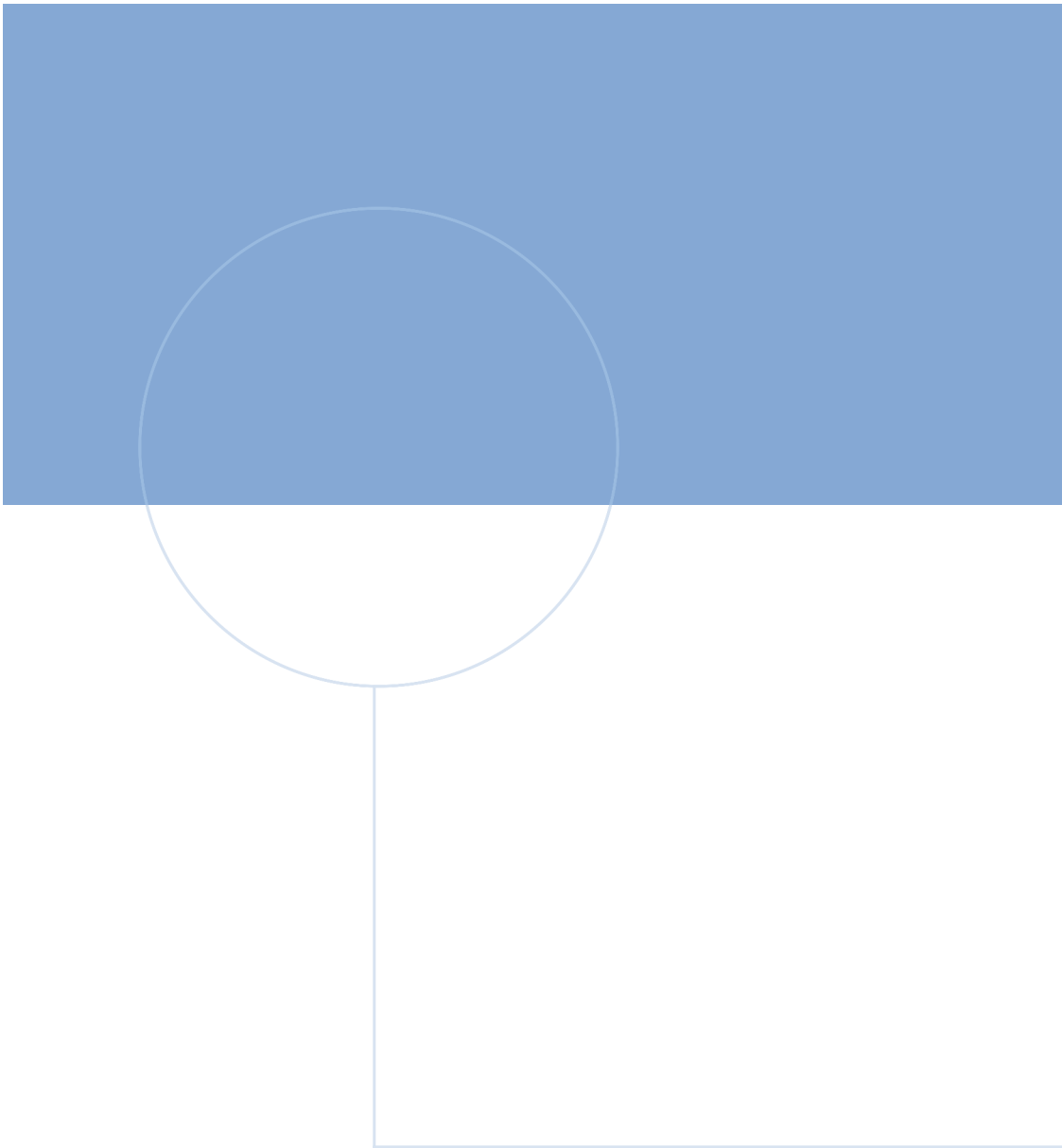
[11] M. Huba, S. Skogestad, M. Fikar, M. Hovd, T. Johansen and B. Rohal-Ilkiv, *Selected Topics on Constrained and Nonlinear Control*. STU/NTNU, 2011, https://folk.ntnu.no/skoge/prost/proceedings/nil-project-book-2011/nil1-tbook.pdf.

[12] H. Seki, S. Ooyama and M. Ogawa, 'Nonlinear model predictive control using successive linearization,' *Transactions of the Society of Instrument and Control Engineers*, vol. 38, pp. 61–66, Jan. 2002. DOI: 10.9746/sicetr1965.38.61.

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

[14] J. P. Jordanou, 'Echo state networks for online learning control and MPC of unknown dynamic systems: Applications in the control of oil wells,' Universidade Federal de Santa Catarina, 2019.

[15] *Casadi docs - optimal control with casadi*, Accessed: 2021-12-7. [Online]. Available: https://web.casadi.org/docs/#document-ocp.



NTNU

Norwegian University of
Science and Technology