

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**  
**CENTRO TECNOLÓGICO**  
**DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS**

**Christopher de Carvalho**

**Internship Report**  
**Obtaining neural network models for prediction**  
**with MPC**

Florianópolis

2020

**Christopher de Carvalho**

# **Obtaining neural network models for prediction with MPC**

Report submitted to Universidade Federal de Santa Catarina as requirement for approval in the subject **DAS 5501 - Estágio em Controle e Automação Industrial** from the Control and Automation Engineering graduation course.

Advisor: Julio Elias Normey-Rico

Florianópolis

2020

Christopher de Carvalho

## Obtaining neural network models for prediction with MPC

This internship report was judged in the context of the subject DAS5501: Estágio em Controle e Automação and was **APPROVED** in its final form by the Control and Automation Engineering course.

Florianópolis, December 18th, 2020

---

**Julio Elias Normey-Rico**

Advisor

Universidade Federal de Santa Catarina

---

**Carolina Maia Vettorazzo**

Local Supervisor

SCoPI

# Resumo

Entre as técnicas de controle avançado, a aplicação de controladores preditivos (MPC) é uma das mais difundidas em sistemas de controle industriais, especialmente na indústria petroquímica. Apesar de a maioria dos processos industriais serem consideravelmente não-lineares, o que se observa na prática é a utilização de controladores preditivos lineares, em conjunto com a aplicação de técnicas de obtenção e linearização de modelos de processos, com o intuito de garantir a resolução do problema de otimização sem exceder o tempo de amostragem. Com a utilização de modelos linearizados, é possível dividir o cálculo da predição de resposta em duas parcelas: a resposta livre e a resposta forçada. Como o cálculo da resposta livre não envolve otimização, é desejável a utilização de um modelo não-linear do processo, para que sejam efetuadas melhores predições. No controlador preditivo atualmente empregado no controle do sistema de compressão de gás de uma plataforma de extração de petróleo da Petrobras, o modelo linearizado do processo também é utilizado para o cálculo da resposta livre. O objetivo desse trabalho é implementar um programa, na linguagem Python, que faça a identificação do sistema de compressão de gás com modelos de redes neurais, a partir de um conjunto de dados do sistema em operação. Além disso, o programa deve ser capaz de exportar os modelos obtidos em um formato compatível com o framework MPA, para que possam ser utilizados pelo controlador preditivo, visando aproveitar-se das vantagens obtidas pela utilização de redes neurais para aumentar o desempenho computacional do sistema de controle.

**Palavras-chave:** identificação de sistemas. redes neurais. MPC. sistemas de compressão de gás

# Abstract

Among the advanced control techniques, the application of predictive controllers (MPC) is one of the most widespread in industrial control systems, especially in the petrochemical industry. Even though most of the industrial processes are highly nonlinear, what is observed in practice is the employment of linear predictive controllers, together with the application of process models obtainment and linearization techniques, in order to guarantee the resolution of the optimisation problem without exceeding the sampling time. With the employment of linearised models, it is possible to split the predicted response calculation in two components: the free response and the forced response. As the calculation of the free response does not involve optimisation, the use of a nonlinear process model is desirable, so that better predictions are made. In the predictive control currently employed in the gas compression system of an oil extraction platform from Petrobras, the linearised model of the process is also used for calculating the free response. The objective of this work is to implement a program, in the Python language, that performs the identification of the gas compression system with neural networks, using a dataset of the real process in operation. In addition, the program should be able of exporting the obtained models in a format compatible with the MPA framework, in order for them to be used with the predictive controller, aiming to improve the computational performance of the control system due to the advantages of using neural networks for simulating the process.

**Key-words:** system identification.neural networks.MPC.gas compression systems

# List of Figures

Figure 1 – Production platform . . . . .	12
Figure 2 – The gas compression system . . . . .	12
Figure 3 – MPC block diagram with neural model for the free response prediction	15
Figure 4 – The Dual-Layer Perceptron . . . . .	19
Figure 5 – The Radial Basis Neural Network . . . . .	19
Figure 6 – The Recurrent Neural Network . . . . .	20
Figure 7 – The cell of a LSTM network . . . . .	21
Figure 8 – The structure of the multi-input single-output neural model of the $m^{th}$ output, $m = 1, \dots, n_y$ . . . . .	22
Figure 9 – The MIMO system represented by $n_y$ DLPs . . . . .	23
Figure 10 – Correct outputs, model predictions and baseline plots . . . . .	26
Figure 11 – Flow chart for the neural identification problem . . . . .	27
Figure 12 – Proposed identification procedure . . . . .	28
Figure 13 – Serial-parallel training configuration . . . . .	29
Figure 14 – Parallel training configuration . . . . .	30
Figure 15 – Different input selection results due to the selected horizon . . . . .	32
Figure 16 – Decision path of the algorithm used in [1] for input selection . . . . .	34
Figure 17 – K selection results for example 2.1 from [2] . . . . .	35
Figure 18 – K selection results for example 2.2 from [2] . . . . .	36
Figure 19 – Dataset ready to be used for training . . . . .	40
Figure 20 – Parameters for the input selection function . . . . .	41
Figure 21 – Default trimmed dataset for the $y_1$ model input selection . . . . .	42
Figure 22 – Possible input selection decision path . . . . .	43
Figure 23 – An instance of <b>regressors</b> . . . . .	43
Figure 24 – X set building example . . . . .	44
Figure 25 – Input selection results dictionary . . . . .	46
Figure 26 – K selection parameters dictionary . . . . .	47
Figure 27 – K selection results dictionary . . . . .	48
Figure 28 – An instance of the <b>analysis dictionary</b> . . . . .	49
Figure 29 – An entry of the <b>model dictionary</b> . . . . .	49
Figure 30 – Single plot example . . . . .	50
Figure 31 – Multiplot for the first four models, trained in serial-parallel . . . . .	51
Figure 32 – One step ahead predictions for $y_2$ model, trained in serial-parallel . . .	52
Figure 33 – <b>analysis dictionary</b> obtained in the first attempt . . . . .	52
Figure 34 – 40 step ahead predictions for $y_2$ model, trained serial-parallel . . . . .	54
Figure 35 – 40 step ahead evaluation of the models trained in serial-parallel . . . .	54

Figure 36 – The validation model . . . . .	55
Figure 37 – Input selection results for the validation model . . . . .	56
Figure 38 – Parameters for the input selection of the validation model . . . . .	57
Figure 39 – Plot of the validation model predictions . . . . .	57
Figure 40 – Zoomed plot of the validation model predictions . . . . .	58
Figure 41 – Analysis dictionary from the second attempt . . . . .	59
Figure 42 – Plot of the $y_2$ model predictions . . . . .	59
Figure 43 – Multiplot of the models obtained in the second attempt . . . . .	60
Figure 44 – Plot evaluations for the models obtained from the simulation dataset .	62
Figure 45 – Plot of the $y_1$ model from the EMSO dataset . . . . .	63
Figure 46 – Zoomed plot of the $y_1$ model from the EMSO dataset . . . . .	63
Figure 47 – Multiplot of the first four models from the simulation model . . . . .	64

# List of abbreviations and acronyms

DLL	Dynamic-Link Library
DLP	Dual-Layer Perceptron
DMC	Dynamic Matrix Control
EMSO	Environment for Modeling, Simulation, and Optimization
FNN	False Nearest Neighbors
GPC	Generalized Predictive Control
GRU	Gated Recurrent Unit
LSTM	Long-Short Term Memory
MIMO	Multiple-Input Multiple-Output
MISO	Multi-Input Single-Output
MLP	Multi-Layer Perceptron
MPA	Módulo de Procedimientos Automatizados (Automated Procedures Module)
MPC	Model Predictive Control
NARX	Nonlinear Autoregressive Exogenous Model
NLP	Natural Language Processing
PID	Proportional–Integral–Derivative
PNMPC	Practical Nonlinear Model Predictive Control
RBF	Radial-Basis Function
RBNN	Radial-Basis Neural Network
RNN	Recurrent Neural Network



# Table of Contents

<b>1</b>	<b>INTRODUCTION . . . . .</b>	<b>10</b>
<b>1.1</b>	<b>The gas compression system . . . . .</b>	<b>11</b>
<b>1.2</b>	<b>The MPC prediction model . . . . .</b>	<b>13</b>
<b>1.3</b>	<b>Project objectives . . . . .</b>	<b>15</b>
<b>1.4</b>	<b>Document organization . . . . .</b>	<b>16</b>
<b>2</b>	<b>THE SCOPI GROUP . . . . .</b>	<b>17</b>
<b>3</b>	<b>NEURAL NETWORKS AS PROCESS MODELS . . . . .</b>	<b>18</b>
<b>3.1</b>	<b>Neural network types . . . . .</b>	<b>18</b>
<b>3.2</b>	<b>The DLP . . . . .</b>	<b>21</b>
<b>3.3</b>	<b>Fitting neural models . . . . .</b>	<b>23</b>
3.3.1	The loss function . . . . .	24
3.3.2	The overfitting problem . . . . .	24
3.3.3	Performance evaluation . . . . .	25
3.3.4	The baseline model . . . . .	25
<b>4</b>	<b>THE IDENTIFICATION PROCEDURE . . . . .</b>	<b>27</b>
<b>4.1</b>	<b>Training configurations . . . . .</b>	<b>29</b>
4.1.1	Serial-parallel . . . . .	29
4.1.2	Parallel . . . . .	30
<b>4.2</b>	<b>Choosing the training configuration . . . . .</b>	<b>31</b>
<b>4.3</b>	<b>The input selection problem . . . . .</b>	<b>32</b>
<b>4.4</b>	<b>The K selection problem . . . . .</b>	<b>34</b>
<b>4.5</b>	<b>Previous iteration . . . . .</b>	<b>37</b>
<b>5</b>	<b>SOFTWARE IMPLEMENTATION . . . . .</b>	<b>38</b>
<b>5.1</b>	<b>Technologies . . . . .</b>	<b>38</b>
<b>5.2</b>	<b>Program structure . . . . .</b>	<b>39</b>
<b>5.3</b>	<b>Flow of execution . . . . .</b>	<b>40</b>
<b>5.4</b>	<b>Input selection algorithm . . . . .</b>	<b>41</b>
<b>5.5</b>	<b>K selection algorithm . . . . .</b>	<b>46</b>
<b>5.6</b>	<b>Analysis Dictionary . . . . .</b>	<b>48</b>
<b>5.7</b>	<b>Model exportation and plotting . . . . .</b>	<b>49</b>
<b>6</b>	<b>RESULTS . . . . .</b>	<b>51</b>
<b>6.1</b>	<b>The first attempt . . . . .</b>	<b>51</b>

6.2	The Simulink validation model . . . . .	55
6.3	The second attempt . . . . .	58
6.4	The simulation dataset . . . . .	61
7	CONCLUSION . . . . .	65
7.1	Future developments . . . . .	65
	REFERENCES . . . . .	67

# 1 Introduction

Model Predictive Control (MPC) emerged in late 70s, and has since become a powerful and practical control technique for industrial processes. The MPC is one of the few advanced control techniques that has achieved significant impact in industrial processes, especially in the petrochemical industry [3].

A couple reasons for that is that the MPC has the ability to handle Multiple-Input for Multiple-Output (MIMO) systems, whether linear or nonlinear, and also because constraints can be imposed on the controlled variables (or output process variables). It can also work with dead-time processes, which are very common in the petrochemical industry.

The MPC calculates the predictions of the future outputs of the process. For doing so, a model of the process is required. With the predictions, an optimisation module calculates the control action that should be applied to the manipulated variables on the next timestep. The optimisation module considers the minimisation of a determined cost function and also the constraints imposed on the variables.

MPC algorithms based on linear models, like the famous Dynamic Matrix Control (DMC), are readily available on commercial softwares that are used in oil refineries around the world, including Petrobras. In these algorithms, the optimisation problem to be solved, at each sampling time, is a quadratic programming (QP) problem, when using a quadratic cost function and linear constraints.

Although advanced controllers based on MPC are usually used in the on-shore industry, most of the off-shore control systems work only with classical control strategies in the regulatory level. The application of advanced modeling and control techniques allows an improvement in the performance of the processes, while avoiding interruptions and allowing greater production, which translates into greater profitability. Thus, several research work have been conducted in the last years to analyse the application of MPC strategies in off-shore oil and gas extraction platforms [4].

A critical system for the functioning of oil and gas extraction platforms is the gas compression system. In cases where this system has to be interrupted, all the process chain may be as well, leading to waste of production and large financial losses. Currently, due to many problems with the control systems in the platforms, unscheduled interruptions have to be made, which impairs their performance. In addition, during periods without interruptions, they still operate far from the ideal efficiency points, having a lower production than what is possible. Furthermore, this system is a complex process with nonlinear dynamics that imposes several challenges to the MPC implementation.

When the process is highly nonlinear or when the operation range changes regularly, the nonlinear model of the process has to be taken into account during the control project phase. For the gas compression system of Petrobras platforms, in particular, a nonlinear MPC algorithm has been used in previous projects, achieving interesting results [5].

From a conceptual point of view, the nonlinear MPC algorithms have the same philosophy as the linear ones. Both can use the same cost function, with the objective of finding the control sequence that minimises this function, while making the predictions in a sliding horizon.

From a practical point of view, it is much harder to calculate the control sequence when working with nonlinear models, because the nonlinear predictions are complex (and time consuming) and the optimisation problem to be solved is not a QP problem, hence nonlinear optimisation methods have to be used. The second problem has been avoided by using linearization techniques, which avoids having to deal with nonlinear optimisation, but still maintains an acceptable performance.

This work investigates the use of neural network as process models for use by the MPC to calculate its predictions. Neural nets can be executed quickly and have good representation ability, therefore being an interesting alternative to the use of nonlinear models, which may be complex to obtain and expensive to simulate in real time operations. Regarding implementation, the obtained models have to be exported to the proprietary software platform MPA, where the MPC is currently executed.

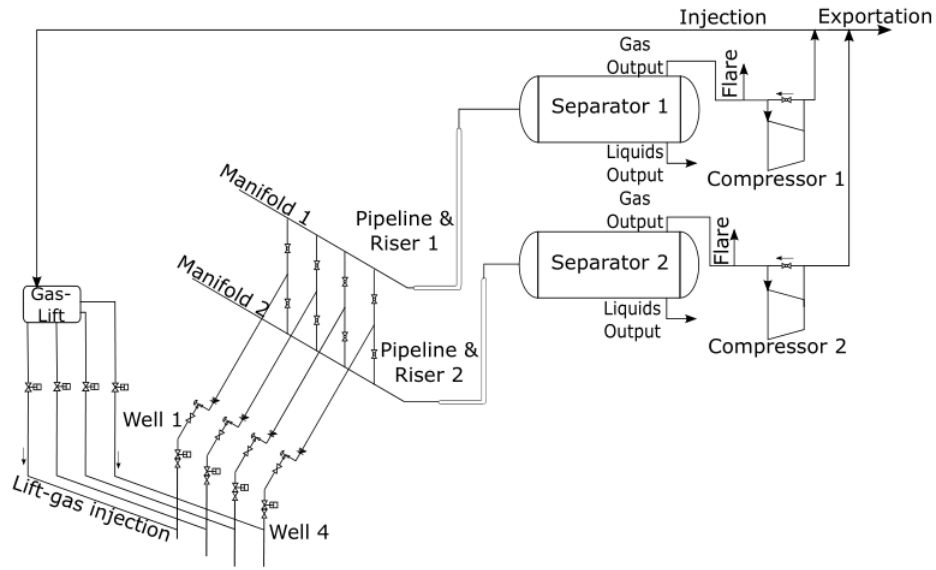
In the sequence, the studied plant is briefly explained, the strategy of splitting the MPC response in two components is presented and the use of neural networks as prediction models for MPC is proposed.

## 1.1 The gas compression system

In an off-shore platform gas compression system, the separators receive a multiphase flow, emanating from the wells. Then, the gas phase is directed to the compressors. According to [4], “In production plants, compressors are the subsystems responsible for giving energy to the gas to increase its pressure, supplying a certain gas flow rate at a specific pressure according to the desired operating point and the specifications of the subsequent subsystem”. The compressed gas coming out of the compressors can be exported or returned to the reservoir in injection wells.

Figure 2 shows a scheme of the different sectors of an off-shore platform, which includes the wells where the fluid is produced, the risers that conducts it to the platform, the separators where the liquid is separated from the gas and the gas compression units.

Figure 1 – Production platform

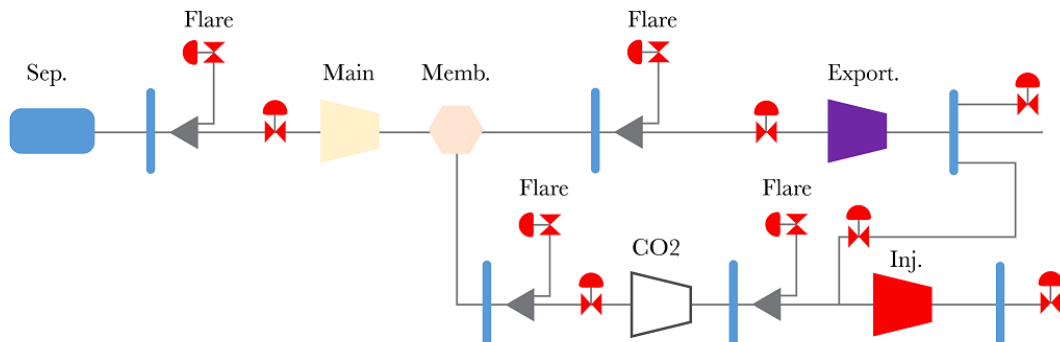


Source: [6]

The gas production system is composed by compressors, scrubbers, heat exchangers, recycling valves and pipelines for making the connections. Its main objective is to compress the gas to be used in the gas injection process or exported for sale.

Figure 2 shows the subject gas compression system of this work, which is composed of three compressors: the main compressor, the exportation compressor and the injection compressor. The CO<sub>2</sub> compressor from the figure should be disregarded. The main compressor has two parallel lines with one compression stage each. The exportation compressor also has two parallel lines, but with two compression stages in each line, and the injection compressor has only one line with one stage.

Figure 2 – The gas compression system



Source: SCoPI project documentation

The main reasons for productivity waste in a compression system is due to the reaching of one of two undesirable operation conditions, namely stonewall and surge conditions. The first occurs when sonic velocity is reached at the exit of a compressor, and the second occurs when the compressor does not have enough input flow to produce sufficient output flow [7]. When one of these conditions is reached, the system has to be interrupted, for safety reasons.

The general control solution for the system is composed of two control layers. The first layer has local control systems, which are responsible for keeping the compressors away from entering dangerous operation zones. Because it is a MIMO system, a MPC operating in the second layer can be very useful for assisting the local controllers by modifying their setpoints, hence driving the process as a whole to more adequate operation points, maximizing production efficiency and reducing energy consumption. The MPC prediction ability helps to avoid reaching undesirable conditions beforehand. Details about the use of MPC in off-shore gas compression systems can be found in [4].

## 1.2 The MPC prediction model

Some time was invested studying the MPC subject. It is an advanced control technique, distinct from the classical control approaches studied in graduation. The MPC takes advantage from the fact of having a (hopefully accurate) model of the process by using this model to predict what will happen in the future, hence taking the necessary actions beforehand.

The majority of industrial processes are considerably nonlinear, including the gas compression system. Its first-principle model consists of a series of differential equations [4], with characteristics resembling those typical seen in fluid mechanics models.

Arguably, the best control solution, in terms of precision, would be to use the nonlinear equation system with a nonlinear MPC algorithm. This approach, however, has considerable disadvantages: the optimisation problem becomes a nonlinear task that must be solved at each sampling instant, therefore, some difficulties may arise. One of them is that the optimisation problem may be non-convex, having many local minima [2]. Trying to guarantee anything, in this scenario, is difficult. Furthermore, an accurate nonlinear model representation is not always available.

According to [8], “Linearization is the only method which has found any wider use in industry beyond demonstration projects. For industry there has to be clear justification for solving nonlinear programs on-line in a dynamic setting and there are no examples to bear that out in a convincing manner.”

MPC algorithms that work with linearised models and perform quadratic optimisa-

tion are called sub-optimal MPC algorithms [2]. Their idea is to circumvent the need of doing nonlinear optimisation. Solving quadratic optimisation problems is computationally less demanding and the convergence is guaranteed, among other desirable properties, from the optimisation point of view.

In the most advanced configurations, the nonlinear model is linearised on-line, for the current operation point. Ideally, this is done at each sampling instant. In simpler configurations, the linear model is obtained only once, off-line, before starting the operation.

One advantage of using linearised models is that linear system properties can be used to split the predicted output response into two components: the forced response and the free response. The forced response component depends on the future control sequence, which is obtained by solving the optimisation problem of minimizing the objective function. The free response component, however, only depends on the past, that is, on the previous control actions and measurements of the system output, all of which is known at the current sampling instant.

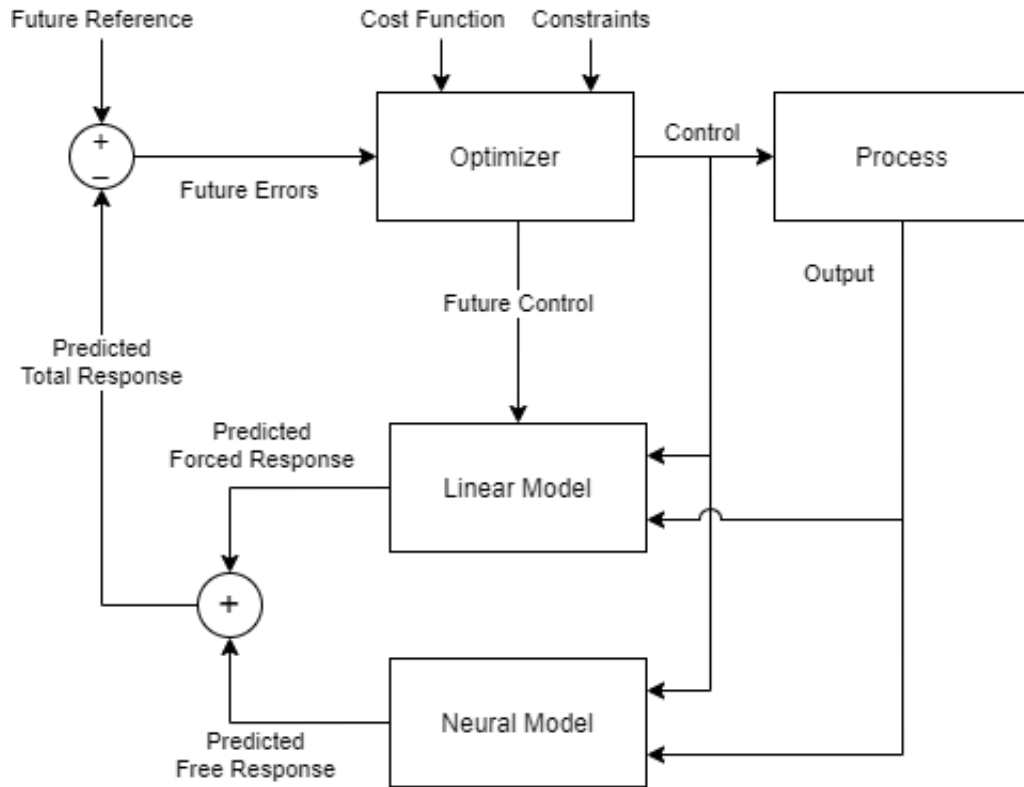
Having split the predicted response into two components, it is then preferable to use a nonlinear model of the process, if available, to calculate the free response component, since it should give more accurate predictions, while the computational burden is not significantly increased by doing so. Neural network models are especially useful for this purpose [2], due to many advantages that come with their use:

- They are precise and have good representation ability;
- They have a relatively small number of parameters that need to be adjusted;
- They can be executed quickly. Unlike fundamental models, neural models do not consist of any differential equations need to be repeatedly solved on-line;
- A great number of efficient neural network building and training open-source platforms are available.

It should be noted that the linearised model would still be used for calculating the forced response component. Figure 3 shows the block diagram implementation of the proposed control strategy.

Neural networks are classified as empirical models [2]. One characteristic of empirical models is that the developer sees the process as a black box (from the input-output perspective), without having to delve into intrinsic modelling details, although having some knowledge of the process can help in result analysis and in the selection of the model input variables.

Figure 3 – MPC block diagram with neural model for the free response prediction



Source: by the Author

The sub-optimal MPC algorithm that is currently being used in the gas compression system is the Practical Nonlinear Model Predictive Control (PNMPC) [9]. Roughly speaking, the idea of the PNMPC is to obtain the linear model by capturing the step response coefficients from simulations of the plant. This is done off-line, for the objective operation point. Currently, the linearised model is used to calculate both components of the response. Using a neural network model for calculating the free response should help to improve the control system's accuracy and computational performance, which serves as motivation for this work.

### 1.3 Project objectives

The main objective of this project is to obtain neural network models that can satisfactorily represent the gas compression system. These models should be identified from a provided dataset, which contains measurements of the real process variables during routine operation. The obtained models would then be used by the PNMPC for calculating the free response predictions, although the PNMPC implementation and tests are out of the scope of this project.



## 1.4 Document organization

This report is organized as follows: in the second chapter, a brief resume of the SCoPI project is presented. In chapter three, the use of neural networks as process models is discussed and the possible neural network structures are presented. In chapter four, the identification procedure is summarized, the possible training configurations are explained and model structure selection algorithms employed in the literature are reported. In chapter five, the used software technologies are listed and the implemented program is explained. In chapter six, the obtained results are analyzed and discussed. Chapter seven gives the conclusion and future perspectives.

## 2 The SCoPI group

This work is part of the project "Development of Non-Linear Predictive Control Algorithms and Performance Evaluation of Predictive Controllers for Oil Production Platforms (Phase 2)", conducted by the SCoPI/DAS/UFSC group. The objective is to make, from this collaboration, the development of advanced control strategies for production platforms employing linear and nonlinear predictive control algorithms, while considering the economic objectives in conjunction with the dynamic performance objectives.

The SCoPI group (from portuguese Software and Control for Industrial Processes) is currently composed by four control and automation engineers and three UFSC professors, in addition to scientific initiation and master's scholarship holders. Despite being made official in 2017, the group has been working on projects in partnership with Petrobras since 2015. The developed activities are conducted by the professor Julio Elias Normey Rico.

## 3 Neural networks as process models

The introductory chapter in [2] gives an overview about different process models used in MPC algorithms. It starts by pointing out desirable properties of a good model: approximation accuracy, physical interpretation, suitability for control and easiness of development. It can be noticed that some of the criteria are contradictory. Each model structure will have its advantages and disadvantages, and an appropriate choice of the model structure may affect the reliability of the control system.

Three general classes of models are available: first-principle, empirical and hybrid models. First-principle models are developed using technological knowledge, and their parameters have physical interpretations. Empirical models have their structure chosen arbitrarily, and their parameters, which have no physical interpretation, can be adjusted using data recorded from the process during operation. Examples of empirical models typically used for nonlinear processes are polynomials, cascade, Fuzzy and Neural Network models. Hybrid models are mixtures of the previous two.

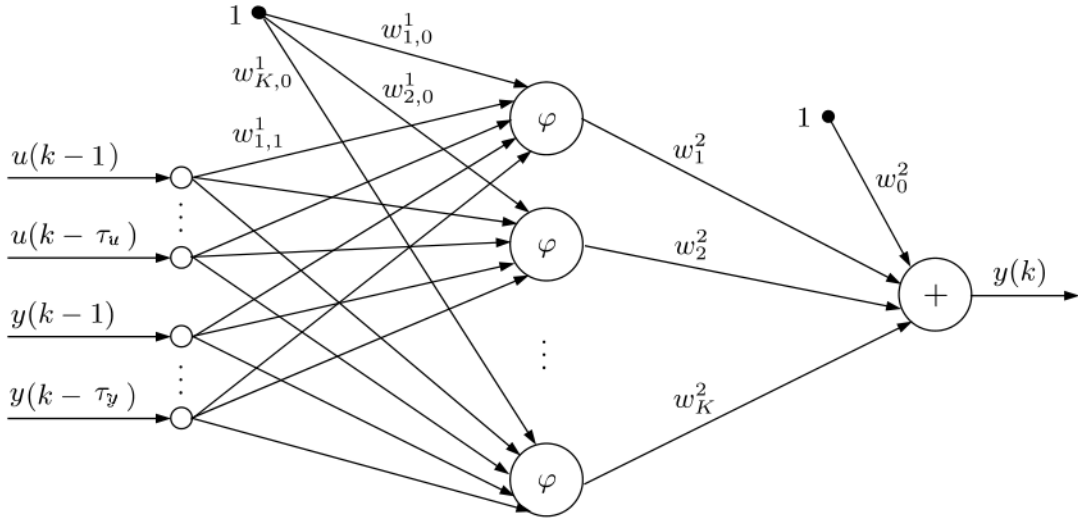
Neural network models have some practical advantages over other models, one of them is the fact that they can be very precise and have good interpolation and extrapolation properties, while having a significantly lower number of parameters [2]. Although they can be used to represent a wide range of processes, using a neural model isn't always the best choice, and different model structures may be better suited for specific processes. For this project, the use of a neural network model was predetermined.

### 3.1 Neural network types

There are many different types of neural networks. The Multi-Layer Perceptron (MLP) feedforward structure is the most popular. For the purpose of representing nonlinear systems, a MLP with two layers is usually used [2], with one layer being the hidden layer and the other the single output layer (the input layer is not counted). This characterizes the Dual-Layer Perceptron (DLP), depicted in Figure 4. The hyperbolic tangent function, represented as  $\varphi$  in the figure, is typically used as the activation function for the hidden nodes, while the identity function is used for the output node.

According to [10], DLP networks are frequently used in practice as models of different processes. This trend was also observed in the studied papers, as it will be seen in Chapter 4. About the representation ability of the DLP, it is mentioned in [11]: "It can be proven theoretically that the MLP network with only one nonlinear hidden layer (i.e. the DLP) is able to approximate any continuous function with an arbitrary degree of

Figure 4 – The Dual-Layer Perceptron

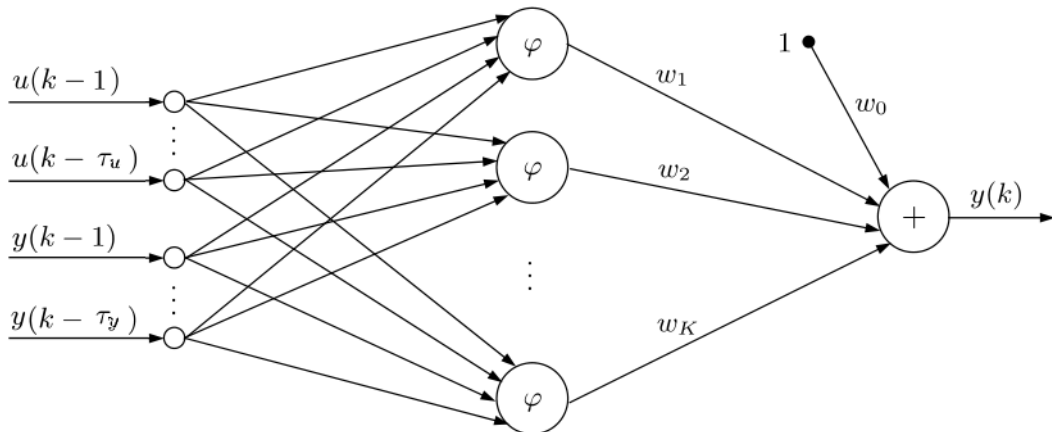


Source: Adapted from [2]

accuracy, provided that the number of hidden nodes is sufficient enough”.

Other network type considered for the project is the Radial Basis Neural Network (RBNN), which is a DLP with Radial Basis Functions (RBF) in the hidden nodes (Figure 5), but it typically has more parameters than a MLP structure of similar accuracy [2]. Also considered was the use of Recurrent Neural Networks (RNN), which allow the outputs of the internal nodes (e.g. the hidden state) to be used as inputs in the next timestep.

Figure 5 – The Radial Basis Neural Network



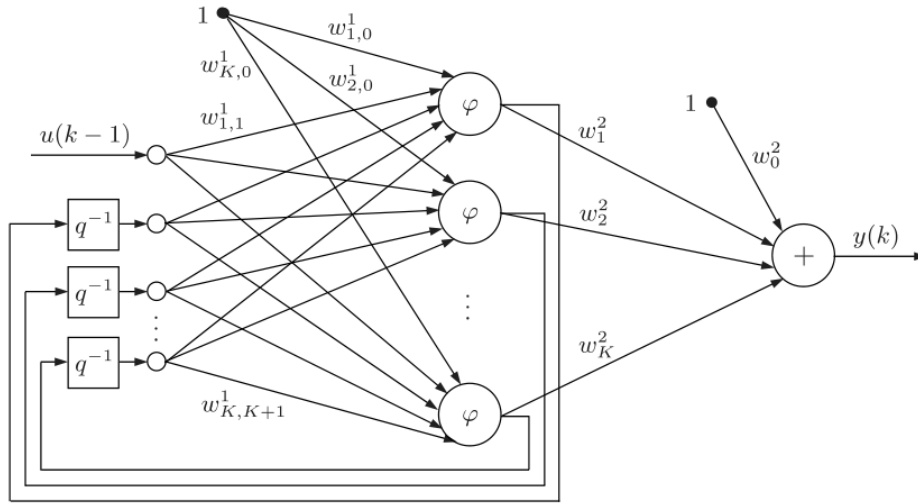
Source: Adapted from [2]

In this project, it was chosen to use the DLP as the neural network type. The plan was to use it first, and if poor results were obtained because of it, then to change

the structure to maybe two layers or move to a different network type. The main reasons for choosing the DLP include: it is a simple architecture, hence being a good option for first endeavors, while also being relatively easy to implement and train. Also, it is a recommended structure in the main book used for study [2], where some examples are provided, and its use is very commonly seen in practice.

In later development stages, the three most common types of Recurrent Neural Networks (RNNs) were also tested, they are the standard RNN (a.k.a. Elman structure), the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU), all of them were studied in [12]. Figure 6 shows the structure of a standard RNN.

Figure 6 – The Recurrent Neural Network



Source: Adapted from [2]

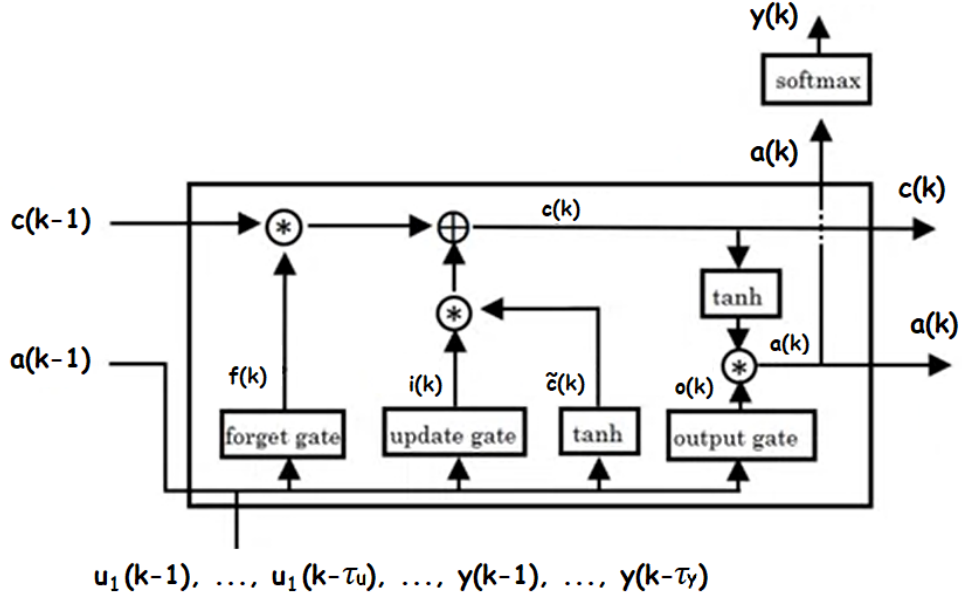
In contrast to the feedforward only MLP, a RNN contains feedback connections from its internal nodes to the inputs. This allows them to exhibit dynamic behavior, being able to make a temporal sequence of predictions. The memory ability granted from passing previous calculations to the next timesteps makes them especially useful for tasks such as Natural Language Processing (NLP) [12].

The standard RNN has only the recursion of the hidden state. The hidden state is a vector composed by the outputs of the hidden layer nodes, and is used as an additional input of the neural network. For the first timestep, an initial state has to be provided, which is usually done by feeding a vector of zeros, although a warm-up procedure can also be performed.

Problems with vanishing and/or exploding gradients, especially in networks with many timesteps (i.e. long prediction horizon), lead to the development of the LSTM networks [13]. Figure 7 shows the basic cell of a LSTM network. The LSTM has two

hidden states, the cell state ( $c$ ) and the hidden state ( $a$ ), which are passed on to the next timestep. Their complex internal structure helps dealing with the gradient problems and gives the LSTM the ability of “remembering” specific features for a longer number of timesteps. This quality proved to be especially useful in NLP problems [12], where the input phrase can be a long sequence of words.

Figure 7 – The cell of a LSTM network



Source: Adapted from [12]

The GRU is an architecture that is relatively recent, being very similar to the LSTM. The difference is the GRU has two gates (reset and update gates), whereas the LSTM has three gates (input, output and forget gates). Using less parameters makes it execute and train faster than the LSTM, at the cost of a little performance drop. Comparisons between the two can be seen in more detail in [12].

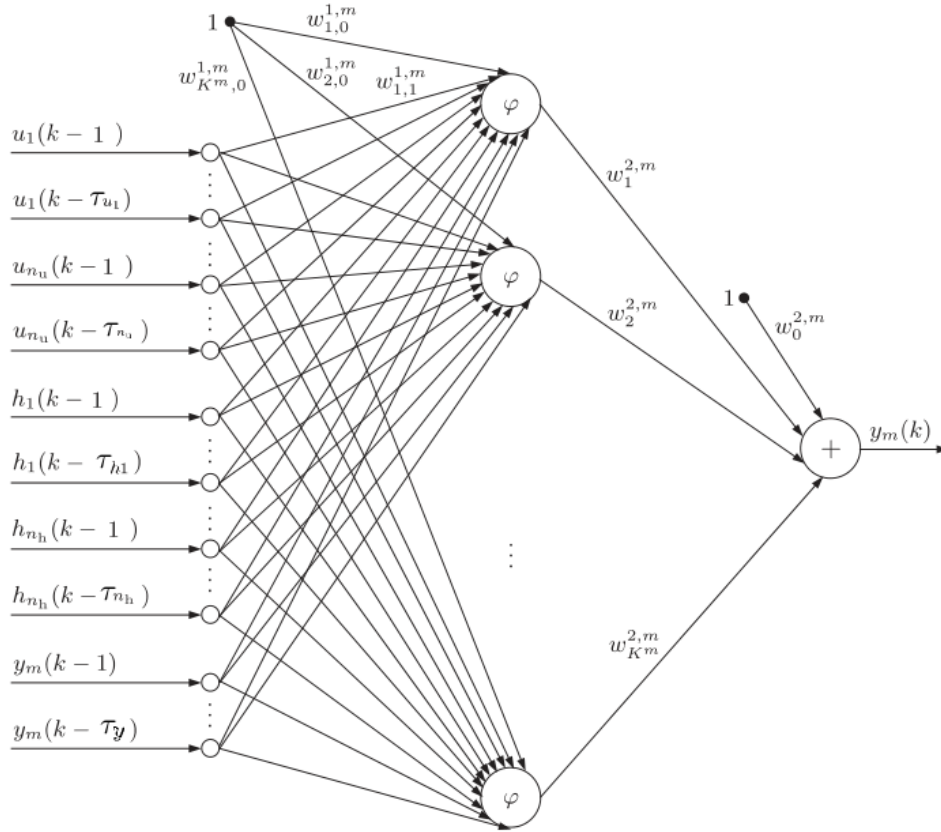
## 3.2 The DLP

It should be noted that the default DLP network is a Multi-Input Single-Output (MISO) system, while the gas compression system is a Multi-Input Multi-Output (MIMO) system, composed of many manipulated and process variables. According to [2], a MIMO system can be satisfactorily represented by  $n_y$  independent neural network MISO models, in this case,  $n_y$  DLPs trained independently.

The structure of the DLP network used for the  $m^{th}$  output model is presented in Figure 8. The network has an input layer with  $I^m$  inputs, one hidden layer containing

$K^m$  nodes using the hyperbolic tangent activation function and one output layer with a single node, which outputs the predicted value for  $y_m(k)$  by a linear sum of its inputs.

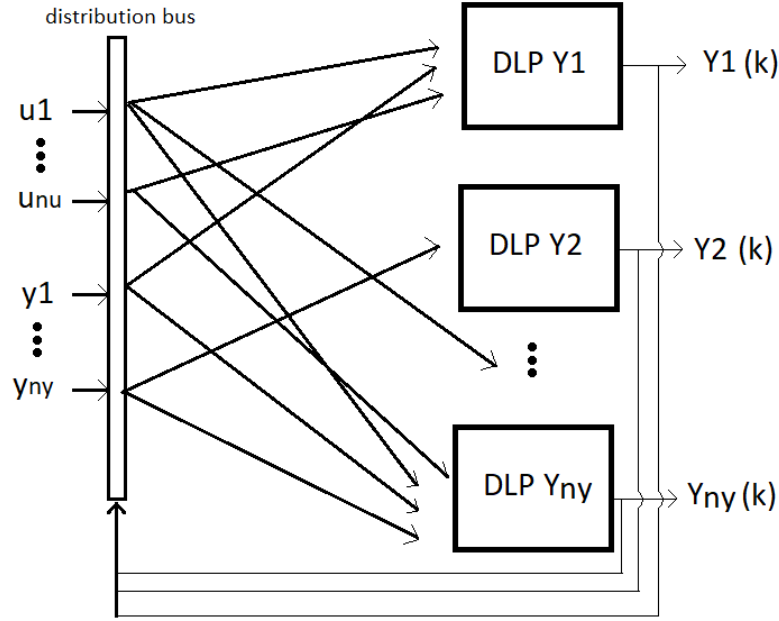
Figure 8 – The structure of the multi-input single-output neural model of the  $m^{th}$  output,  $m = 1, \dots, n_y$



Source: Adapted from [2]

The model inputs are composed by the manipulated variables  $u_1, \dots, u_{n_u}$ , the measured disturbances  $h_1, \dots, h_{n_h}$ , which in this case are considered to be the other output variables, and by the past values of the output  $y_m$ .

Figure 9 depicts how the MIMO system can be emulated from the combination of  $n_y$  MISO neural networks. In the left side of the figure, there is the representation of a bus which is responsible for receiving the measurements of all process variables (including the networks predictions) and distributing them accordingly to the requirements of each internal network.

Figure 9 – The MIMO system represented by  $n_y$  DLPs

Source: by the author

It should be pointed out that it is possible to try building a MIMO neural network to represent the whole system. This approach was tried on a previous iteration of the same project [14]. However, it has significant disadvantages: in a MIMO MLP network, all the outputs would depend on the same input arguments, whereas in practice, the order of dynamics of consecutive parts of the process is different, therefore, each output has its own group of variables of influence. These nonlinear dependencies would have to be cancelled or at least diminished by the optimisation algorithm. Furthermore, a MIMO network would require much more parameters, which makes the execution of training and testing procedures to be more difficult.

### 3.3 Fitting neural models

Training neural network models, also known as model fitting, is the process of using the dataset to update the model weights in order to create a good mapping of the inputs to the output. A good model should be able to generalise well, that is, mimic the process behavior. In the project's case, this is especially required near the usual operation points of the gas compression process.

Before starting the training procedure, the dataset is divided into sets  $\mathbf{X}$  and  $\mathbf{Y}$ . Each measured sample from the dataset is considered an example, having the respective entry in  $\mathbf{X}$  and  $\mathbf{Y}$ .  $\mathbf{X}$  contains the input variables of the model for a given sample, while  $\mathbf{Y}$



(also known as the labels vector) contains the correct output for that sample.

The model weights are initialized randomly. Each example set of inputs is feed-forwarded through the network, generating a predicted output that results in a prediction error when compared to the respective example of the labels vector. The learning process happens with the minimization of a loss function, which takes into account the errors from all examples. This is done by an optimization algorithm, which back-propagates the derivatives of the loss function through the network connections, updating all the weights.

Each time that the entire training set passes through the network, it is considered an epoch. The default procedure is to update the model weights once per epoch, but it is also possible to do it in fewer examples, in order for speeding-up the learning process. This can be adjusted via a parameter called batch size. The number of epochs for which a model is trained can be adjusted dynamically, via a regularization technique known as early stop, which will be explained in section 3.3.2.

### 3.3.1 The loss function

For the selection of the loss function to be minimised, there are three functions that are commonly chosen for system identification procedures: Sum of Squared Errors (SSE), Mean of Squared Errors (MSE) and Mean of Absolute Errors (MAE). In the used machine learning framework, the MSE and MAE functions are available by default, while the SSE function can be implemented by creating a custom loss function. In the project, the three functions were experimented with, all of them giving very similar results. In the end, the MSE loss function was chosen to be minimised during training, although the SSE performance of the models is also calculated in the analysis phase and shown in the plots.

### 3.3.2 The overfitting problem

It is important to differentiate between a model with good generalisation ability and an overfitted model. The training procedure of neural networks is not trivial, and fitting a model for more epochs than necessary may recur in overfit. Overfitting happens when the model starts to memorize the training set rather than learning useful information from it. To prevent overfitting and improve generalization ability, regularization techniques can be employed.

One regularization technique that was used for all model fitting procedures in this project is the early stopping. The idea is that, during training, as soon as the model performance starts to degrade, or even if the loss begins to increase, then the training procedure is interrupted to avoid overfitting.

Other regularization techniques that were not used for this project include the dropout rate, data augmentation and L1/L2 regularization. Details about these can be

found in [12].

### 3.3.3 Performance evaluation

To estimate the performance of a model, a common practice is to evaluate its performance on a dataset that was not used for training. One technique, known as Train-Validation split, is to divide the dataset into two uneven parts: the training set and the validation set. The training set is used for training the model (i.e., adjusting the weights), while the validation set is used to evaluate the model, typically after a training epoch is completed. It is important to reemphasize that no weight adjustments are made from the loss values obtained with the validation set.

One problem of employing the Train-Validation split is that, even though the validation set is used only for assessment, the model ends up being indirectly biased towards this set, therefore, the performance achieved on it doesn't truly reflect the model's generalisation ability. Arguably, a better solution is to split the dataset into three sets instead of two, namely training, validation and test sets. This is known as the Train-Validation-Test split. The train and validation sets are used in the exact same manner as in a two-split, but the test set is preserved to be used only **after** training is complete (i.e. the model has its final weights), in order to evaluate the model on its final state.

The idea is that test set contains data that the model has never seen before, hence giving an unbiased performance assessment. If the model has good results in the validation set, but poor results in the test set, the most probable cause would be that the model has indeed overfitted to the validation set. Then, an error analysis procedure has to be carried out to find the possible causes and decide what to do. This is a broad topic whose details can be found in [12].

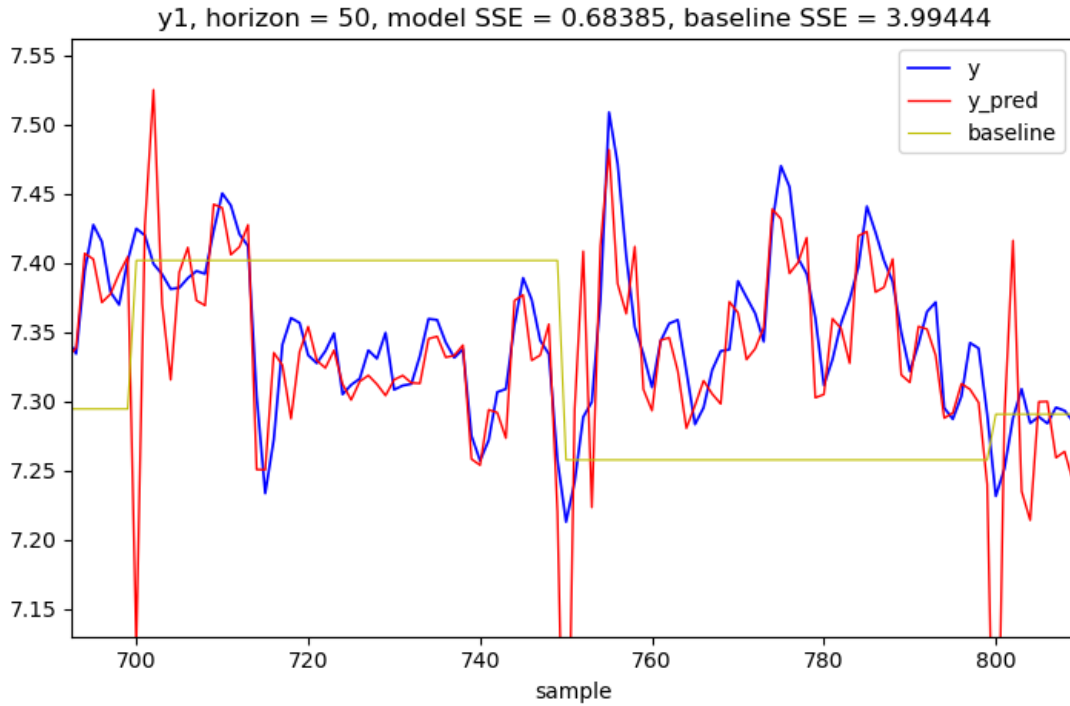
### 3.3.4 The baseline model

An additional way of evaluating a model is by comparing the achieved performance with a baseline performance. Simple baselines commonly used are the linear model response, or simply the sustain of the initial output state by using a Zero-order hold. The latter is used in this project. Figure 10 shows an example of the comparison of the model predictions with the baseline values obtained with a Zero-order hold. In the case, the initial output value is sustained for 50 samples, which is the chosen prediction horizon for the plot. In the next window, the baseline value is updated.

It should be noted that the chosen baseline value is a minimal effort solution, so it is expected that the obtained models should have a considerably better performance.

In the case of this project, a possible reason for achieving a performance worse

Figure 10 – Correct outputs, model predictions and baseline plots



Source: by the Author

than baseline is that the chosen input variables orders were incorrect. Then, one course of action would be to redo the input selection procedure using different parameters.

In this chapter, the advantages of using neural networks as process models were discussed. The considered network structures were presented and the selection of the DLP network for first endeavors was justified. Then, the idea of representing the MIMO system with  $n_y$  DLPs was proposed, and finally the topic of fitting neural models was covered.

In the next chapter, the identification procedure with neural networks is discussed, together with the proposed identification steps. Then, the possible training configurations are presented and a literature investigation is conducted to find which solutions have been used in practice.

## 4 The identification procedure

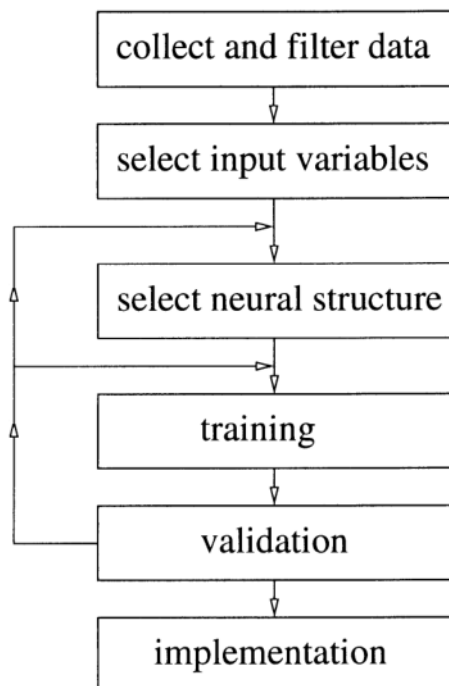
In [2], the identification procedure of a neural model is summarized in three steps:

1. Selection of the model structure: choice of model inputs and the number of hidden nodes;
2. Model training: optimisation of model weights;
3. Model validation: assessment of the generalization ability of the trained model.

It is suggested that training a series of neural models and choosing the best one is a simple, but in practice quite an effective approach, although specialised algorithms can be used to determine the dynamic order of the models.

Figure 11 illustrates the standard steps of an identification procedure, including the data collection and implementation steps. For the DLP, the only neural structure selection that can be made is to determine the number of nodes in the hidden layer.

Figure 11 – Flow chart for the neural identification problem

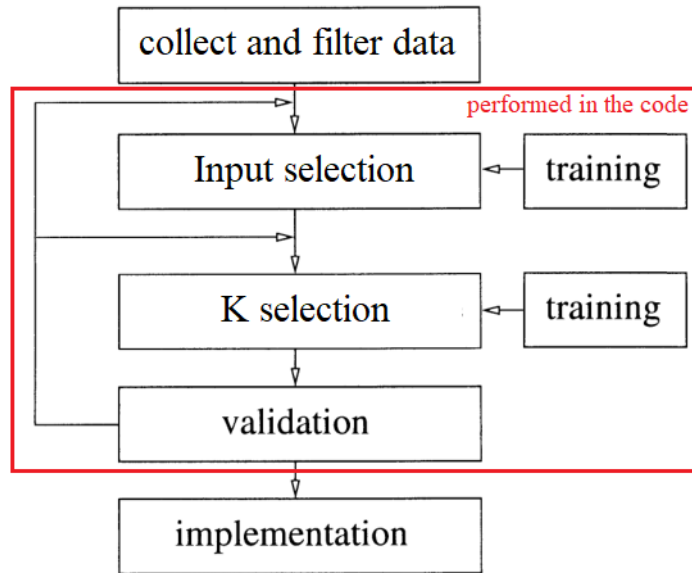


Source: [15]

In order to take advantage that all model structure selection is performed in the same code, the identification procedure is structured differently for this project (Figure 12),

in such a way that model training and evaluation are involved in both steps of the selection procedures, being used for verifying the best performing structures. In addition, the optimised (final) model is returned from the K selection function, which selects the number of nodes in the hidden layer.

Figure 12 – Proposed identification procedure



Source: Adapted from [15]

There are three steps that can be resumed as follows:

- **Input selection:** the choice of model inputs is done empirically by an algorithm that selects the best order for each input from a predefined set of input variables. This is done by using a growth strategy, where the considered order, for each input, starts at zero and increases through the stages. Model training and evaluation are used by the algorithm to make its decision path;
- **K selection:** with the model inputs defined, another algorithm searches for the best number of nodes for the hidden layer, here referred to by the capital letter “K”. Model training and evaluation are also involved in this algorithm’s decision process. The model to be returned from this function already has its final parameters;
- **Model validation:** The obtained model is evaluated on the whole dataset and also on the test set (if a three-way split was used). The predictions are plotted to give a visual feedback of the model performance. Baseline values (3.3.4) can also be plotted and used for comparison. In addition, useful information about the trained models is concatenated in a dictionary structure, enabling the user to quickly investigate the obtained results.

Other people's work that served as inspiration for the elaboration of the model structure selection algorithms are presented and discussed in sections 4.3 and 4.4.

## 4.1 Training configurations

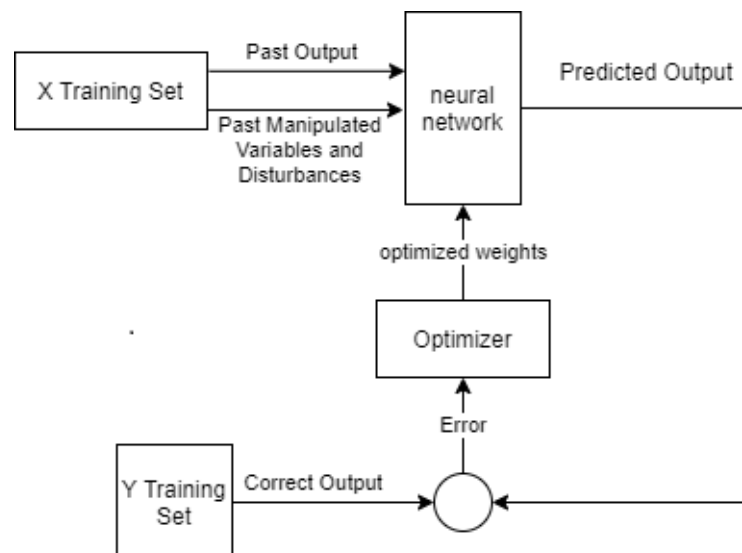
During training, two model configurations are possible: serial-parallel (a.k.a. non-recurrent) and parallel. The training configuration is a parameter that must be defined by the developer before implementing the code, and will be used for the whole program execution, whenever a model is trained or evaluated.

It should be noted that the chosen training configuration is relevant only during model training and evaluation stages. For instance, after deployment, two models trained in different configurations, but with the **same** structure and parameters, would be used in the exact same way and would have the same results.

### 4.1.1 Serial-parallel

In the non-recurrent configuration, the output signal is a function of the process input and output signal values from previous sampling instants that come exclusively from the X training set. So, for training purposes, each sampling instant extracted from the dataset into the training sets can be treated as an individual example.

Figure 13 – Serial-parallel training configuration



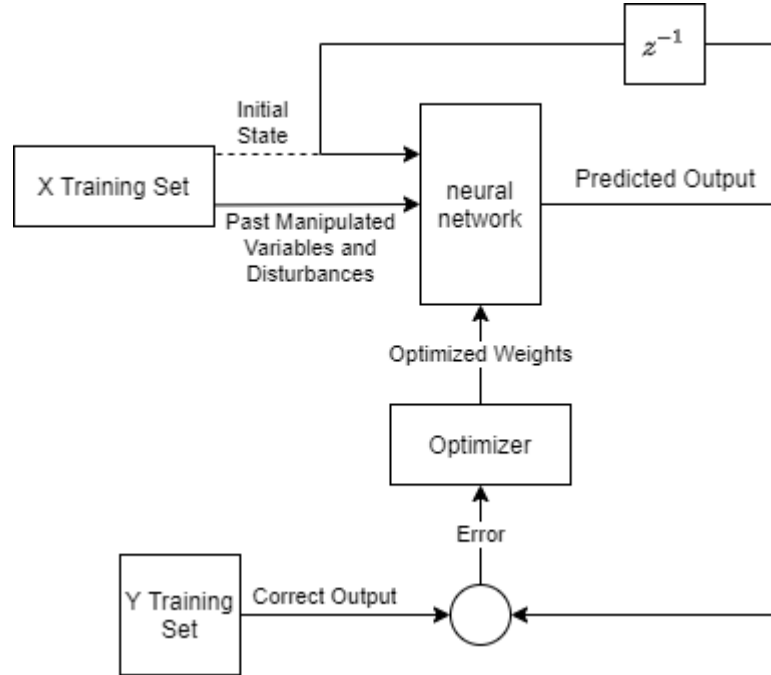
Source: Adapted from [16]

Using the serial-parallel configuration is equivalent to using a prediction horizon equal to one for making the predictions. For each sampling instant, a prediction is made, and then, in the following timestep, the previous value for the output is extracted from the X training set, rather than using the predicted output value from the previous timestep.

#### 4.1.2 Parallel

In the Parallel configuration, the inputs respective to the past output values are replaced by the model predictions from previous sampling instants. So, the difference in this configuration is that the predicted values are recurrently used as inputs for the next time steps. For the first prediction, an initial state has to be provided. During training, the sum of the model errors over the whole prediction horizon is minimized.

Figure 14 – Parallel training configuration



Source: Adapted from [16]

The parallel configuration is equivalent to using a prediction horizon greater than one. It is possible to use an horizon up to the number of examples that can be extracted from the dataset, but a better option is to use the same horizon (or maybe a bit longer) that the MPC will use for its predictions. Choosing a very large horizon favours the occurrence of vanishing/exploding gradient problems, because in this configuration, the Back-Propagation Through Time (BPTT) algorithm is used for parameters optimisation, hence the derivatives have to travel through the whole prediction window.

The TensorFlow framework, which was used for performing all model fitting procedures, has not a default class for creating models with output feedback at each timestep, hence, to use the parallel configuration, it is necessary to create a custom model class. In TensorFlow's documentation, a tutorial is available, which guides this kind of implementation [17].

## 4.2 Choosing the training configuration

In [2], it is pointed out that training in the parallel configuration is much more computationally demanding in comparison with training a classical serial-parallel model, although it is later stated that the training for all examples in the book are carried out using the parallel (recurrent) configuration, indicating the author's preference for it.

In [16], it is adverted that “if the parallel structure is employed, it cannot be guaranteed that the learning process of the weights will converge or the error between the output of the system and that of the network will tend to zero”. In addition, in [18] it stated: “There are also situations in which this type of network (recurrent parallel) is not capable of capturing the whole plant state information of the modeled process. The use of real plant outputs avoids many of the encountered analytical difficulties, assures stability and simplifies the identification procedure. This type of feedforward network is known as a series-parallel model”.

The arguments from the previous paragraph's made me choose the serial-parallel configuration for initial endeavors. In addition, in this configuration, each sampling instant can be analyzed independently, so the dataset shuffling technique (very commonly used) can be employed, aiming to improve the model generalization ability. In the parallel configuration, this technique is harder to be implemented effectively, because it is necessary to keep track of the consecutive samples on each prediction window example.

However, halfway through the development, the analysis of preliminary results suggested that the serial-parallel configuration was not ideal, hence the code was changed to allow the use of the parallel configuration. The justification is that, when using the serial-parallel configuration, which is analogue to using a prediction horizon of one, the previous value of the output becomes a great bet for the output of the next timestep. This makes the model put too much weight in the previous value of the output, confusing the input selection algorithm.

In the serial-parallel configuration, the correct previous value of the output are fed to the network at every timestep. This is a problem, because the model, after deployment, will not receive the correct previous output values at every instant, instead it has to use the values predicted by itself. So, training with the parallel configuration is the best option.



In [15], a neural model is used for modelling a solar plant and later used with a Generalized Predictive Control (GPC). The idea is familiar: use the linear model to calculate the forced response and the nonlinear neural model to calculate the free response. The general problem of nonlinear system identification is presented, and the use of an NARX model realized by neural networks is proposed. To prevent problems during training, all variables are normalized. This is a standard procedure in training neural networks.

Also in [15], there is an interesting experiment that gives evidence of how changing the prediction horizon may modify the results of an input selection algorithm. Figure 15 contains the resulting table from this experiment. The four numbers in the Model input column correspond to the considered order for the input variables, being, in order, the manipulated variable, the process variable and two measured disturbances.

Figure 15 – Different input selection results due to the selected horizon

SSE in the VS for the one-step prediction ( $h = 1$ ) and the recursive 25-step-ahead prediction ( $h = 25$ ) for all models that use a total of eight input variables. The columns on the left correspond to linear models, and the ones on the right to neural networks

Model input	Linear models		Non-linear	
	$h = 1$	$h = 25$	$h = 1$	$h = 25$
1151	0.531	10.1	0.345	8.01
1241	0.089	25.3	0.076	15.0
1331	0.057	13.2	0.045	14.1
1421	0.056	15.1	0.042	12.3
1511	0.055	14.4	0.039	10.1
2141	0.431	7.21	0.331	6.21
2231	0.049	9.44	0.028	9.02
2321	0.036	4.31	0.030	5.04
2411	0.036	6.12	0.031	5.23
3131	0.344	4.80	0.291	4.70
3221	0.049	9.20	0.026	8.91
3311	<b>0.032</b>	4.51	0.029	3.20
<b>4121</b>	0.291	<b>3.80</b>	<b>0.025</b>	2.96
4211	0.052	12.0	0.301	9.04
<b>5111</b>	0.270	4.03	0.092	<b>2.90</b>

Source: Adapted from [15]

An interesting observation is that the best one-step-ahead predictor does not yields the best 25 step ahead predictions. The explanation given in the paper is that, for a one step ahead prediction, the variable that best represents the output  $y(k)$  of the model is  $y(k-1)$ , but this value is affected by noises which are propagated to successive predictions.

### 4.3 The input selection problem

For each output, a neural model is created, and for each model, the selection of which variables to consider as inputs, as well as what order to consider for each one, was one of the main problems investigated. In the proposed identification procedure, this is

done in the input selection step. An investigation of the literature was performed to find out what has been done in practice to solve this problem.

In [19], two mathematical methods are presented: Lipschitz Numbers and False Nearest Neighbors (FNN). These methods make use of only input-output data for achieving a NARX model representation of the process. They could be resorted to as a last resort, because implementing them, for each output, would be a complex task to be done, considering that the gas compression system is a MIMO system.

In [20], a neural network is trained and validated to represent a heat exchanger mesh. Then, a predictive controller is designed and tuned based on the neural network for temperature control. A DLP network is used, with eight neurons in the hidden layer and one neuron in the output layer. The order for the model inputs is derived directly from the available first-principle model, and the number of hidden nodes  $K$  is selected empirically. The training performance is assessed by the MSE loss function.

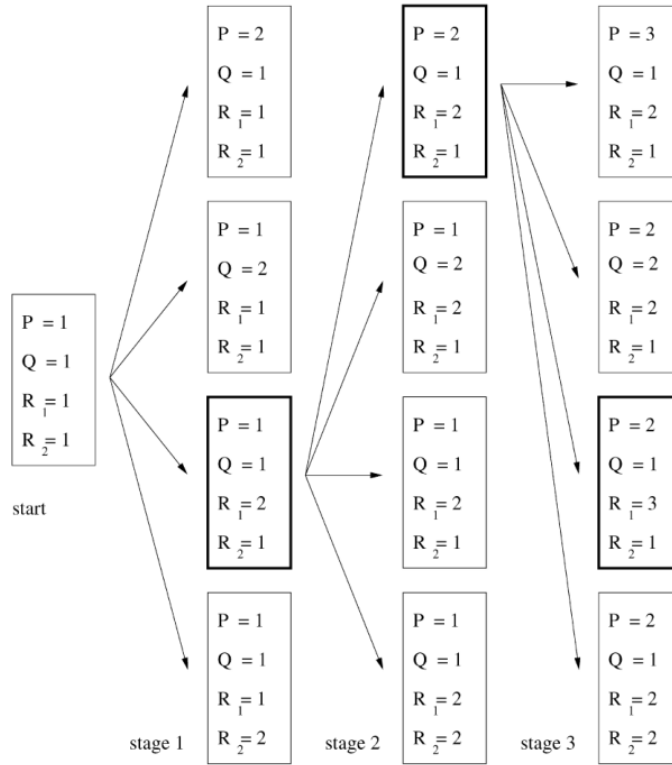
In [1], it is shown the construction of a neural model for a solar plant, using collected data. This model is able to be updated online, this is done by using a sub-network with the objective of countering small discrepancies and temporal changes. The solar power plant is a MISO system. For selection of the network's inputs, some methods are discussed, including the already mentioned FNN, but the chosen method is one which was originally proposed in [21], by one of the authors. This method is based on the gradients, and starts by considering only the  $(k - 1)$  past values of the process variables ( $u$ ,  $y$  and the disturbances), increasing them as necessary.

The graph with the algorithm decision path for consecutive stages presented in [1] is shown in Figure 16. The variables  $P$ ,  $Q$ ,  $R_1$  and  $R_2$  represent the considered order for the model inputs of the manipulated variable, the process variable and the two measured disturbances, respectively. At each stage, there are four possible options of increasing the order of one of the inputs. Each option is tested, and the one that generates the greatest performance gain is chosen for the next stage, until a stop condition is reached.

This decision strategy was chosen to be used for this project's input selection algorithm. The difference is that, in this project, the decision of which option to choose is made by training and evaluating neural networks that have the considered inputs, while in the paper, this was done by using the mathematical method proposed by one of the authors.

The algorithm implemented in this project starts with a model that has no inputs considered, apart from the  $y(k - 1)$  input that is necessary to have for maintaining the recursive aspect. In the first stage, the algorithm considers increasing the order of each input variable independently, and chooses the option which has the best performance. Each option is tested for a pre-defined number of times, and the mean loss is taken. This is done

Figure 16 – Decision path of the algorithm used in [1] for input selection



Source: [1]

because the problem is nonlinear and the weights are initialized randomly, so considering the average loss, for each option, helps achieving more consistent results. Then, the mean losses obtained by each option are compared for deciding which the best option to choose. It should be noted that the number of nodes in the hidden layer is kept constant for this procedure, set to a reasonable value. The algorithm stops if one of the stopping criteria is reached. Further implementation details are presented in section 5.4.

In [18], it is stated that the problem of input selection for neural network models is not satisfactorily solved yet. If the order of the process is known, all necessary past inputs and outputs should be fed to the network, which can make the input space become large. In many cases, there is no possibility of learning the order of the model, and the number of suitable delays has to be selected experimentally, which was the case for this project.

#### 4.4 The K selection problem

The algorithm that selects the number of nodes in the hidden layer was exclusively based on the examples provided in the second chapter of [2], where two example systems are identified and used to compare some proposed neural network based MPC algorithms. The

systems are: a yeast fermentation reactor and a high-pressure high-purity ethylene-ethane distillation column. The identification procedure used for these examples are presented here, in particular the method for choosing the number of hidden layer nodes  $K$ .

In the yeast fermentation reactor process, the first-principle model is known, consisting of some nonlinear differential equations, which is treated as the real process during the procedure. Aiming to make the neural model have a good generalization ability, the dataset is divided into three parts: training, validation (also known as development) and test sets. After each training iteration, the model error for the validation set is calculated. When the error increases, training is ceased to prevent overfitting.

Each set has 4000 samples that resulted from open-loop simulation of the first-principle model. The sampling time is 30 minutes, and the output signal contains a small measurement noise. A DLP is used for modelling, with the hyperbolic tangent as the activation function for the hidden layer and the identity function for the single output node. Process variables are scaled, but not normalized. The accuracy of the model is assessed using the SSE loss function.

To decide the number of inputs, a set of models with different inputs is trained, each one containing a sufficiently high number of hidden nodes, the case,  $K$  was set to 10. It is found out empirically that the model should have second-order dynamics.

Having selected the inputs, a test is made to decide the proper number of hidden nodes  $K$ . Each model configuration, from  $K = 1$  up to  $K = 7$ , is trained 10 times, initialized randomly, and the best results are kept.  $K = 3$  is found out to achieve the lowest SSE in the validation set, so it is selected and evaluated on the test set, having similar results. The linear model of the process is also used for comparison, although its performance is considerably inferior. Figure 17 shows the comparison of the results. “NP” corresponds to the number of parameters of the models.

Figure 17 –  $K$  selection results for example 2.1 from [2]

Model	NP	SSE <sub>train</sub>	SSE <sub>val</sub>	SSE <sub>test</sub>
Linear	4	$9.1815 \times 10^1$	$7.7787 \times 10^1$	–
Neural, $K = 1$	7	$1.1649 \times 10^1$	$1.3895 \times 10^1$	–
Neural, $K = 2$	13	$3.2821 \times 10^{-1}$	$3.2568 \times 10^{-1}$	–
Neural, $K = 3$	19	$2.0137 \times 10^{-1}$	$1.8273 \times 10^{-1}$	$1.4682 \times 10^{-1}$
Neural, $K = 4$	25	$1.9868 \times 10^{-1}$	$1.9063 \times 10^{-1}$	–
Neural, $K = 5$	31	$1.3642 \times 10^{-1}$	$1.9712 \times 10^{-1}$	–
Neural, $K = 6$	37	$1.3404 \times 10^{-1}$	$2.0440 \times 10^{-1}$	–
Neural, $K = 7$	43	$1.2801 \times 10^{-1}$	$2.9391 \times 10^{-1}$	–

Source: Adapted from [2]

The second example system is a distillation column used in a polish refinery. The plant has three PID controllers in the basic control layer, responsible for the stabilisation of the flow rate and ensuring safe process operation. The MPC is used in the supervisory layer for process optimisation and to reduce energy consumption. The control system architecture is reminiscent of the one used for the gas compression system.

The dynamic first-principle model of the process is available, but it consists of around 17000 differential nonlinear equation, which is too complicated to be used directly in the MPC algorithm. A steady-state model is available, and, from it, in addition with recorded time-responses of the process, a simplified dynamic model is defined, having the serial Hammerstein structure. During simulations, this model is treated as the real process.

For the inputs selection, the solution was to simply use the same order as the simplified first-principle model:  $y(k) = f(u(k-3), y(k-1))$ . For the K selection, an experiment is conducted in the same way as it was done for the first example. It is found out that using  $K = 5$  gives the best results. This structure was evaluated in the test set, having a successful performance. Figure 18 shows the table with the structures comparison.

Figure 18 – K selection results for example 2.2 from [2]

Model	NP	SSE <sub>train</sub>	SSE <sub>val</sub>	SSE <sub>test</sub>
Linear	2	$2.3230 \times 10^1$	$1.6099 \times 10^1$	–
Neural, $K = 1$	5	$1.4855 \times 10^0$	$1.6528 \times 10^0$	–
Neural, $K = 2$	9	$9.4224 \times 10^{-3}$	$1.1372 \times 10^{-2}$	–
Neural, $K = 3$	13	$4.6645 \times 10^{-3}$	$5.3305 \times 10^{-3}$	–
Neural, $K = 4$	17	$3.7614 \times 10^{-3}$	$4.3407 \times 10^{-3}$	–
Neural, $K = 5$	21	$3.1962 \times 10^{-3}$	$3.4611 \times 10^{-3}$	$3.4669 \times 10^{-3}$
Neural, $K = 6$	25	$3.1483 \times 10^{-3}$	$3.3515 \times 10^{-3}$	–
Neural, $K = 7$	29	$3.1316 \times 10^{-3}$	$3.4612 \times 10^{-3}$	–
Neural, $K = 8$	33	$3.0909 \times 10^{-3}$	$3.4819 \times 10^{-3}$	–

Source: Adapted from [2]

After obtaining the neural model, the different proposed neural network based MPC algorithms are tested. For this example, the most complex sub-optimal MPC algorithm manages to achieve a performance comparable to the nonlinear optimisation algorithm.

The K selection algorithm implemented in this project follows the same idea used in these two examples: start with a reasonably small number of hidden nodes and go increasing until performance stops to improve significantly. Being the DLP a relatively small network, it makes sense to use a growing algorithm. Implementation details of the K selection algorithm are given in section 5.5.

## 4.5 Previous iteration

In the previous iteration of the project [14], the MIMO neural network approach was used. For the model structure selection, an interesting pruning algorithm was experimented with, which did both input selection and structure selection procedures as the model was trained. This algorithm is called Neural-FROLS and was withdrawn from [22]. In bigger networks, it makes sense to use pruning techniques, but in smaller networks, like the DLP, growing algorithms fit well.

The results were good for the training data, which was obtained by simulation of the gas compression system. However, the performance with real process data was poor. Possible reasons it could be that the model was not trained using real data (wrong target), or the MIMO network was too big and failed to generalize well for the many inputs, even with the use of a pruning algorithm.

For the current iteration of the project, the approach of using MISO DLP networks for representing the MIMO process was chosen based on recommendations from [2] and also because it was verified to be commonly used in practice, as seen in the studied papers.

In the next chapter, details about the software implementation are presented, including the list of the employed technologies, the program structure, the flow of execution and the implementation of the selection algorithms.

## 5 Software Implementation

### 5.1 Technologies

The model identification program was developed in the Python language. The open-source Anaconda distribution was used for package management and deployment. Anaconda has a great number of data-science related packages. It also allows the use of package environments, so that each project being developed has its own environment of pertinent libraries, which can be easily managed by the developer.

At the beginning of the development, the latest version of Python, 3.8.5, which already came installed in the root environment, was not supported by TensorFlow. So, a separate environment was created for the project, using Python 3.7.9, which was then supported.

Spyder, the utilized IDE, was also installed via Anaconda. Spyder is a scientific environment for Python development, being commonly used by data scientists. It has a plugin called Variable Explorer, which facilitates the inspection of environment variables and has proven to be very useful for this project. The main Python libraries used for the code implementation were:

- TensorFlow: library for machine learning, with a particular focus on developing deep neural networks;
- NumPy: essential library for scientific computing in Python, providing a multidimensional array object, among many other routines and objects;
- Pandas: library for data manipulation and analysis, offers the useful Series and Data Frame structures, which are tabular structures with labeled axes;
- scikit-learn: machine learning library projected to interact with NumPy, being useful for performing data splits, shuffling and normalization;
- SciPy : library for mathematics, science and engineering. In this project, used only to open Matlab `.mat` files;
- Pickle: module for saving and loading Python objects in `.pickle` files;
- Matplotlib: library for creating plots in Python.

Of the mentioned libraries, TensorFlow is the most important, because it provides all the necessary tools for creating, training and evaluating neural networks in Python. In

the beginning of the development, another library, called Keras, was used for that purpose. Keras is a deep learning API that is arguably simpler to use than TensorFlow, as it is more high-level oriented and user-friendly. At cost of flexibility, it allows rapid prototyping to quickly build and test neural networks within a few lines of code.

It happens, however, that as of mid 2017, Keras was fully adopted and integrated into TensorFlow through a module called `tf.keras`. This module allows the user to define models using the friendly Keras interface, while maintaining access to TensorFlow's lower level functionalities when a custom feature is needed. `tf.keras` is also better maintained by the TensorFlow team than Keras. The library `tensorflow-gpu`, which supports running computations on a GPU, was also used for speeding up training.

## 5.2 Program structure

The developed program is divided in six Python files for better organization:

- `main.py`: serves as the starting and ending point for program execution, performing calls to the other modules. It also has an execution dictionary where the user can set all the parameters for training, plotting and analysis;
- `data_utils.py`: contains functions for saving and loading data and files from different sources. Also has data processing functions for trimming, shuffling and splitting the dataset according to the considered input variables and prediction horizon;
- `training_utils.py`: contains the training routines for the two steps of the model creation procedure: `input_selection`, where the model inputs are defined, and `K_selection`, where the number of hidden nodes is defined;
- `custom_models.py`: contains a few different custom classes implementation of neural networks, using the `tf.keras` API and following TensorFlow's guide [17];
- `analysis_utils.py`: controls the analysis procedure that can be performed after training is complete. Has functions to extract useful information from the `training` dictionary and concatenate it in an `analysis` dictionary to be returned;
- `plot_utils.py`: allows the creation of single plots and/or multiplots of the trained models responses. Can also perform baseline comparisons and plotting.

The code has guide comments in portuguese and is available on GitHub [23]. In the following sections, the implementation of the algorithms for model structure selection and analysis will be detailed. The rest of the program consists of normal code development solutions that can be viewed in the source code.



### 5.3 Flow of execution

Before starting the execution, the user may configure, in the main module, the execution dictionary `exec_cfg`, which contains all the necessary parameters for the program execution, including the definition of the group of outputs for which models will be created in that particular execution, so that not all models have to be created at once. This makes it possible to analyze intermediate results.

The training dataset file is loaded via the `load_data` function, which can be found in the `data_utils` module. In future uses, new datasets may come in different ways and formats, so this function has to be adjusted accordingly. For the training procedure, it is required that the dataset array is put in a Pandas Data Frame object, with columns indicating the manipulated variables followed by the output variables, as shown in Figure 19. The example set has four manipulated variables and four process variables.

Figure 19 – Dataset ready to be used for training

Index	u1	u2	u3	u4	y1	y2	y3	y4
0	18.9701	19.3771	45.4417	44.0229	393.951	10092.5	431.098	10227
1	18.9731	19.3805	45.3883	43.9296	394.521	10096.7	431.198	10227
2	18.9761	19.384	45.335	43.8362	395.091	10100.9	431.298	10227
3	18.979	19.3874	45.2816	43.7429	395.661	10105.1	431.398	10227
4	18.982	19.3908	45.2283	43.6496	396.231	10109.3	431.498	10227
5	18.985	19.3943	45.175	43.5562	396.801	10113.5	431.598	10227
6	18.988	19.3977	45.1217	43.4629	397.371	10117.7	431.698	10227
7	18.991	19.4011	45.0683	43.3696	397.941	10121.9	431.798	10227
8	18.9939	19.4046	45.015	43.2762	398.511	10126.1	431.898	10227
9	18.9969	19.408	44.9616	43.1829	399.081	10130.3	431.998	10227
10	18.9999	19.4115	44.9083	43.0896	399.651	10134.5	432.098	10227
11	19.0029	19.4149	44.855	42.9962	400.221	10138.7	432.198	10227
12	19.0058	19.4183	44.8017	42.9029	400.791	10142.9	432.298	10227

Source: by the author

In a loop cycle, a model is created for each output in the range chosen by the user in `exec_cfg`. For each model, first the input selection function is called, which returns the selected input variables and their respective orders in a Pandas Series structure called `regressors`. A dictionary containing training information, called `search results`, is also returned for insertion in the entry for the respective output in the `training dictionary`.

Then, the K selection function is called, which selects the best number of nodes

for the hidden layer, receiving as input the `regressors` found in the previous step. This function returns the definitive model to be used for predictions and also returns information about the training procedure for insertion in the `training dictionary`.

After the model creation loop is completed, the `analysis dictionary` is created. It contains useful information, extracted from the `training dictionary`, which is then organized in a modular structure to facilitate inspections. The created models can then be used for making single plots and/or multiplots. The parameters of the models are stored in another dictionary, called `model dictionary`, that can be later transformed, by an external program, into an XML file for usage in the MPA control platform.

It should be noted that, in order for the analysis and plotting to be performed, not all the models have to be created. The user can choose, for example, to create models for the first three outputs, then perform the analysis and plotting procedures to verify the intermediate results and check if everything is working as expected.

## 5.4 Input selection algorithm

The input selection algorithm is responsible for selecting which process variables to consider as the inputs for the model being created. It receives the dataset, the current output and a parameter dictionary, called `input selection params`.

The parameters dictionary, depicted in Figure 20, is nested into the `exec_cfg` dictionary, which is located in `main.py`. Some of the parameters will be explained together with the algorithm, others are self-explanatory. The three last items correspond to the stopping conditions. If they are set to false, as in the figure, the only stopping condition for the algorithm is when all the stages are covered.

Figure 20 – Parameters for the input selection function

```
"input selection params" : {  
    "max stages": 6,  
    "trains per option": 1,  
    "search patience": 2,  
    "max epochs": 1000,  
    "early stop patience": 3,  
    "hidden layer nodes": 8,  
    "horizon" : 100,  
    "starting y order" : 2,  
    "resampling factor": 1,  
    "validation size": 0.3,  
    "target loss": False,  
    "acceptable loss": False,  
    "min delta loss": False  
},
```

Source: by the author

As mentioned, the dataset should be in a Data Frame object, as exemplified in Figure 19. Before the input selection starts, an additional step can be performed, which is to remove from the dataset the variables which are known to not influence the current output. The function responsible for doing that is the `trim_data` function, which receives a `dependency` list containing the variables to consider as inputs for the input selection of the current model. If no `dependency` list is provided, the default procedure is to assume that the output depends only on the manipulated variables  $u$  and on previous values of itself.

Figure 21 exemplifies a possible returned dataset from the `trim_data` function, when no `dependency` list is informed. In this example, the current output is  $y_1$ , so it is maintained alongside the manipulated variables  $u_1, \dots, u_4$ , while the variables corresponding to the other outputs were removed.

Figure 21 – Default trimmed dataset for the  $y_1$  model input selection

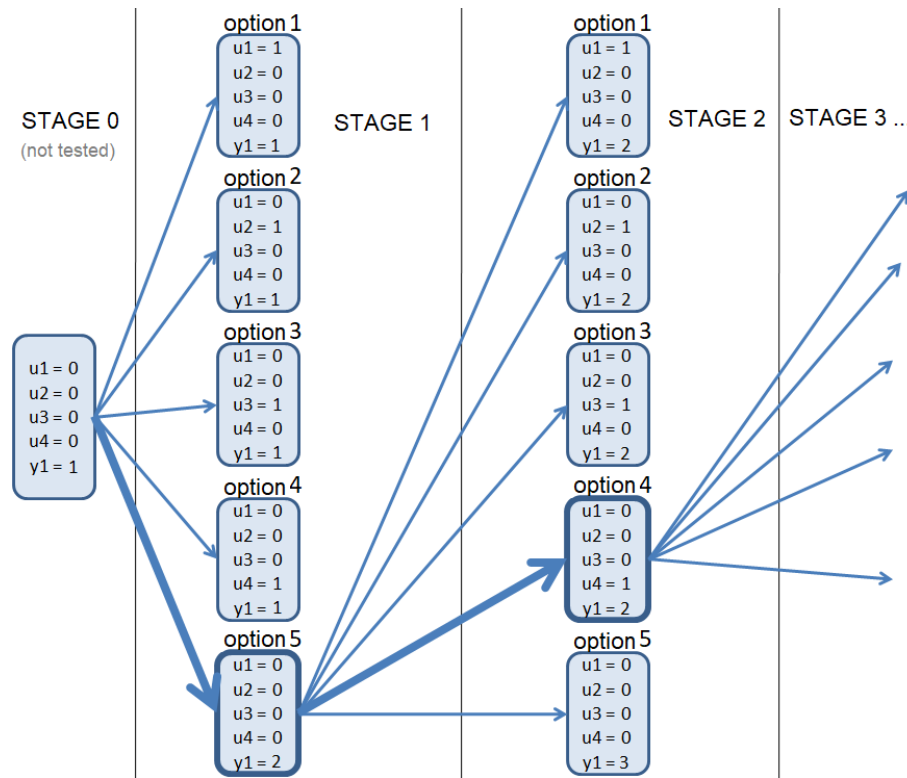
Index	u1	u2	u3	u4	y1
0	18.9701	19.3771	45.4417	44.0229	393.951
1	18.9731	19.3805	45.3883	43.9296	394.521
2	18.9761	19.384	45.335	43.8362	395.091
3	18.979	19.3874	45.2816	43.7429	395.661
4	18.982	19.3908	45.2283	43.6496	396.231
5	18.985	19.3943	45.175	43.5562	396.801
6	18.988	19.3977	45.1217	43.4629	397.371
7	18.991	19.4011	45.0683	43.3696	397.941
8	18.9939	19.4046	45.015	43.2762	398.511
9	18.9969	19.408	44.9616	43.1829	399.081
10	18.9999	19.4115	44.9083	43.0896	399.651
11	19.0029	19.4149	44.855	42.9962	400.221
12	19.0058	19.4183	44.8017	42.9029	400.791

Source: by the author

Alternatively, the `trim_data` function call can be skipped (by altering a parameter in `exec_cfg`). Then, all the variables of the system, including the other outputs, will be considered as possible inputs in the input selection procedure for the current model.

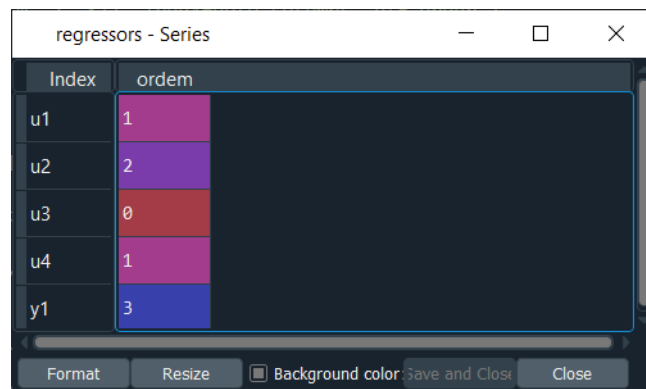
As mentioned in section 4.3, the structure of the input selection algorithm is based on the algorithm used in [15]. Figure 22 exemplifies a possible decision path for the first three stages. In that case, the input data was trimmed to contain the manipulated variables from  $u_1$  to  $u_4$  and the output  $y_1$ . To maintain the recursive aspect for using the parallel training configuration, the order of input  $y_1$  begins at 1, so  $y_1(k-1)$  is selected by default.

Figure 22 – Possible input selection decision path



Source: by the author

The rectangles represent the tested **regressors** for each option. **regressors** is a Pandas Series object that contains the considered order for the input variables of a model. At each stage, the +1 order increment for each input is considered as an option. The option which achieves the best performance, that is, the lowest average MSE loss, is selected for the next stage. This selection is represented by a thick blue arrow.

Figure 23 – An instance of **regressors**

Source: by the author

Figure 23 exemplifies an instance of **regressors**, which contains the variables and their respective orders that should be used as inputs of the model. The exemplified values mean that the model would receive as inputs  $u_1(k-1)$ ,  $u_2(k-1)$ ,  $u_2(k-2)$ ,  $u_4(k-1)$ ,  $y_1(k-1)$ ,  $y_1(k-2)$  and  $y_1(k-3)$ .

In the way the algorithm is currently implemented, it is not possible to consider the  $(k-2)$  value of an input without including the  $(k-1)$  value. The consequence is that possible delays in the inputs have to be cancelled internally by the network's weights optimisation.

It should be noted that the dataset needs to be further prepared in order for being used with a model. This is done in two steps. In the first step, a function called **build\_sets** receives the dataset and the **regressors**, then normalizes the dataset and builds the X and Y arrays.

Figure 24 shows the result of applying the **regressors** from Figure 23 to the dataset from Figure 21 for building the X array. First, the dataset is normalized, then the **regressors** values are taken into consideration, so that each possible example is extracted from the dataset and the synchrony between the samples is kept.

Figure 24 – X set building example

Index	u1(k-1)	u2(k-1)	u2(k-2)	u4(k-1)	y1(k-1)	y1(k-2)	y1(k-3)
0	0.779243	0.747922	0.745646	1	0.186228	0.182294	0.17836
1	0.781205	0.75019	0.747922	0.997714	0.190162	0.186228	0.182294
2	0.783167	0.752465	0.75019	0.995429	0.194095	0.190162	0.186228
3	0.785129	0.754741	0.752465	0.993143	0.19803	0.194095	0.190162
4	0.787091	0.757016	0.754741	0.990858	0.201964	0.19803	0.194095
5	0.789053	0.759284	0.757016	0.988572	0.205897	0.201964	0.19803
6	0.791015	0.76156	0.759284	0.986287	0.209831	0.205897	0.201964
7	0.792977	0.763835	0.76156	0.984001	0.213765	0.209831	0.205897
8	0.794939	0.76611	0.763835	0.981716	0.217699	0.213765	0.209831
9	0.796902	0.768379	0.76611	0.97943	0.221633	0.217699	0.213765
10	0.798864	0.770654	0.768379	0.977145	0.225567	0.221633	0.217699
11	0.799943	0.771897	0.770654	0.976283	0.229501	0.225567	0.221633

Source: by the author

The same procedure is done for obtaining the labels vector Y, which have only one column. The second step in the data preparation is performed by the **recursive\_sets**

function, which receives the X set, the Y (label) set and the chosen prediction horizon. This function is responsible for separating the samples into a list of sequences that have the same number of timesteps as the prediction horizon value. In addition, the sets can be further split into train/validation (two-split) or train/validation/test sets (three-split). By default, the input selection algorithm uses a two-split, while the K selection uses a three-split. The splitting ratios depends on the parameters defined by the user.

For instance, if the prediction horizon is 50 and the dataset has 100k samples, first the `build_sets` function splits the dataset into sets X and Y, having 100k synchronized samples each, then the `recursive_sets` function reorganizes the sets into lists containing 2000 example sequences of 50 samplings each.

The calling of `recursive_sets` is only necessary if using the parallel training configuration, because RNNs require three dimensional inputs for training. The source code [23] provides further information regarding these two data preparation steps.

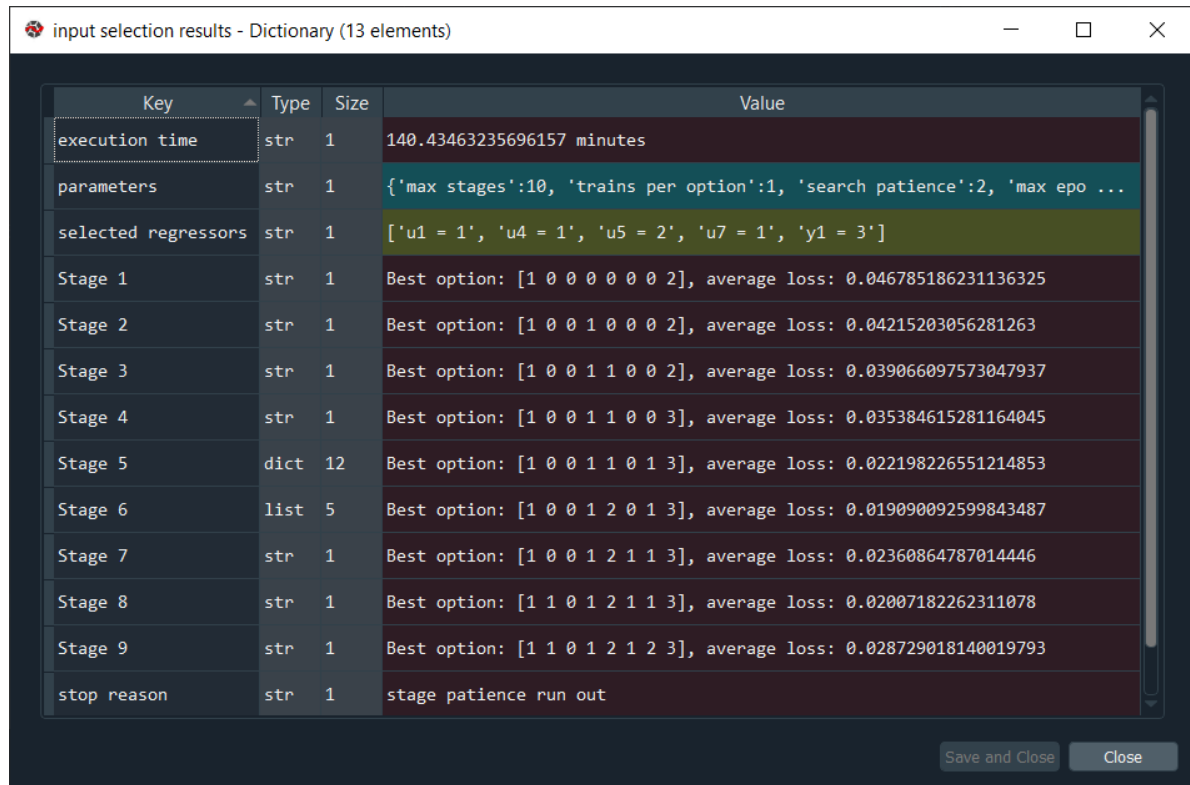
To rank each option, a model containing as inputs the respective value for the `regressors` is initialized, trained and evaluated. This step is repeated a few times, and the average loss, for all the initializations of the model, is calculated. This is done because the weights are initialized randomly, so taking the average loss helps achieving more consistent results. The models have a fixed number of nodes in the hidden layer, which is set to a reasonable value. An early-stop callback is used, so training is automatically ceased when no improvement is detected for consecutive epochs. The search is interrupted when one of the following stop conditions is reached:

- All stages were covered;
- (if informed) Target average loss was reached;
- (if informed) No considerable performance improvement and the acceptable average loss was reached;
- (if informed) No considerable performance improvement and search patience run out.

At the end of the search, an instance of `regressors` is returned, containing the selected inputs, and the `input_selection_results` dictionary for insertion on the `training` dictionary.

Figure 25 shows an example of the `input_selection_results` dictionary. It contains information such as: the parameters used, the stop reason, the execution time, the selected `regressors` and the average loss obtained in the best option of each stage. The algorithm's decision path can also be identified.

Figure 25 – Input selection results dictionary



Key	Type	Size	Value
execution time	str	1	140.43463235696157 minutes
parameters	str	1	{'max stages':10, 'trains per option':1, 'search patience':2, 'max epo ...
selected regressors	str	1	['u1 = 1', 'u4 = 1', 'u5 = 2', 'u7 = 1', 'y1 = 3']
Stage 1	str	1	Best option: [1 0 0 0 0 0 2], average loss: 0.046785186231136325
Stage 2	str	1	Best option: [1 0 0 1 0 0 2], average loss: 0.04215203056281263
Stage 3	str	1	Best option: [1 0 0 1 1 0 2], average loss: 0.039066097573047937
Stage 4	str	1	Best option: [1 0 0 1 1 0 3], average loss: 0.035384615281164045
Stage 5	dict	12	Best option: [1 0 0 1 1 0 1 3], average loss: 0.022198226551214853
Stage 6	list	5	Best option: [1 0 0 1 2 0 1 3], average loss: 0.019090092599843487
Stage 7	str	1	Best option: [1 0 0 1 2 1 1 3], average loss: 0.02360864787014446
Stage 8	str	1	Best option: [1 1 0 1 2 1 1 3], average loss: 0.02007182262311078
Stage 9	str	1	Best option: [1 1 0 1 2 1 2 3], average loss: 0.028729018140019793
stop reason	str	1	stage patience run out

Source: by the author

## 5.5 K selection algorithm

The K selection algorithm is similar to the input selection algorithm in the way that model creation, training and evaluation are used for choosing the best structures. As mentioned in section 4.4, the algorithm is based on the selection procedure applied in the two examples presented in chapter 2 of [2].

“K” is the letter used to refer to the number of nodes in the hidden layer of the DLP, which are also called hidden nodes. The K selection function is called after the input selection and receives the trimmed (or not) dataset, the chosen **regressors** for the model and the execution parameters.

In the input selection algorithm, the **build\_sets** and **recursive\_sets** functions are called many times, because the value for the **regressors** is changed in every option. In the K selection, these functions are called only once, at the beginning of the procedure, because the inputs of the model are already selected, so the sets only need to be created once.

The K selection algorithm is the last step involving model training in the code. The returned model has its final weights. If the test set size is specified in the parameters,

the best model obtained is evaluated on the test set, before being saved.

Figure 26 – K selection parameters dictionary

```
"K selection params" : {  
  "K min": 6,  
  "K max": 10,  
  "trains per K": 3,  
  "search patience": 2,  
  "max epochs": 1000,  
  "early stop patience": 3,  
  "horizon" : 50,  
  "resampling factor": 1,  
  "validation size": 0.2,  
  "test size": 0.15,  
  "target loss": False,  
  "min delta loss": False  
}
```

Source: by the author

Figure 26 shows the parameters dictionary for the K selection procedure, it can be noticed that some parameters are repeated from the input selection parameters. The algorithm starts by creating a model with  $K = Kmin$ . The weights are initialized and the model is trained and evaluated on the validation set. This step is repeated “**trains per K**” times, and the weights and loss of the best performing initialization are kept.

In the next stage, the value for K is incremented in +1 and the procedure is repeated. If the best initialization (for this K) achieves an improvement in the performance, the overall **best K** is value updated to the current K, together with the **best loss** and the weights of the best model. The search continues until one of the following the stopping criteria is reached:

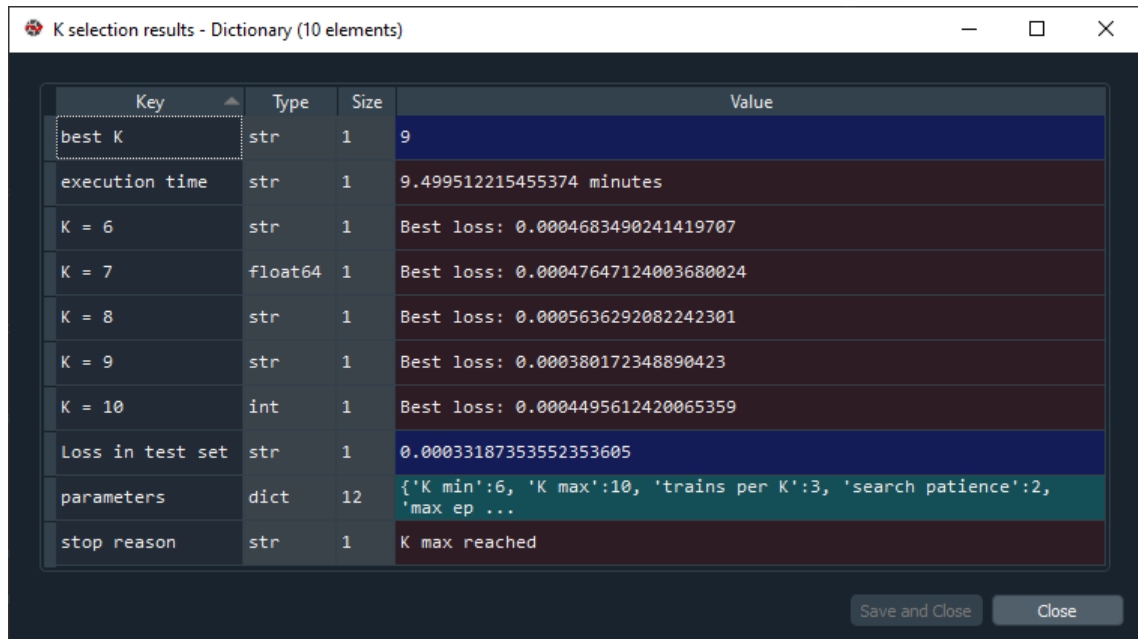
- All K values, within the provided range, were covered;
- (if informed) Target loss value was reached;
- (if informed) No considerable performance improvement and search patience has run out.

At the end of the search, the best value for K and the respective weights, along with the **K selection results** dictionary, are returned to be saved in the entry for the current output on the **training dictionary**.

Figure 27 shows an instance of the **K selection results** dictionary. Just like the **input selection results**, returned by the input selection function, it contains information about the used parameters and the achieved results, including the performance of the best initialized model for each K.



Figure 27 – K selection results dictionary



Key	Type	Size	Value
best K	str	1	9
execution time	str	1	9.499512215455374 minutes
K = 6	str	1	Best loss: 0.0004683490241419707
K = 7	float64	1	Best loss: 0.00047647124003680024
K = 8	str	1	Best loss: 0.0005636292082242301
K = 9	str	1	Best loss: 0.000380172348890423
K = 10	int	1	Best loss: 0.0004495612420065359
Loss in test set	str	1	0.00033187353552353605
parameters	dict	12	{'K min':6, 'K max':10, 'trains per K':3, 'search patience':2, 'max ep ...
stop reason	str	1	K max reached

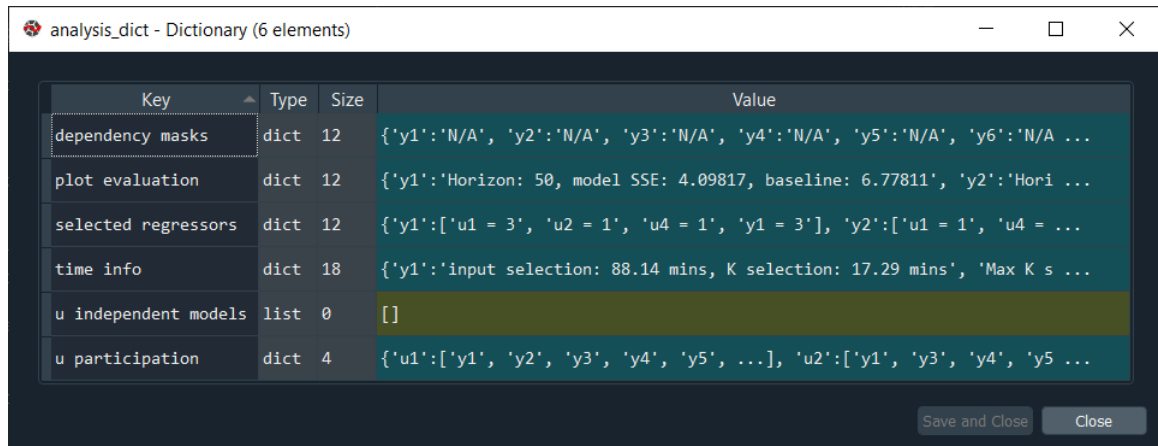
Source: by the author

## 5.6 Analysis Dictionary

After at least one model is created, the `analysis dictionary` can be obtained through the `run_analysis` function. This function receives the `training dictionary` and then proceeds to extract and group useful information from it.

In the `training dictionary`, the entries are organized by the outputs, while in the `analysis dictionary`, the entries are organized by information. Figure 28 shows an instance of the `analysis dictionary`. Each entry contains the following information:

- `u independent models`: indicates which models do not have any manipulated variable `u` in the selected `regressors`, which is undesirable;
- `plot evaluation`: contains the comparison between the models and baselines performances in the single plots;
- `selected regressors`: groups the selected `regressors` for each created model;
- `time info`: contains information about the employed time for building each model, as well as the average, maximum and minimum execution times of the algorithms;
- `u participation`: lists on which models each manipulated variable `u` participates;
- `dependency masks`: groups the informed `dependency masks`.

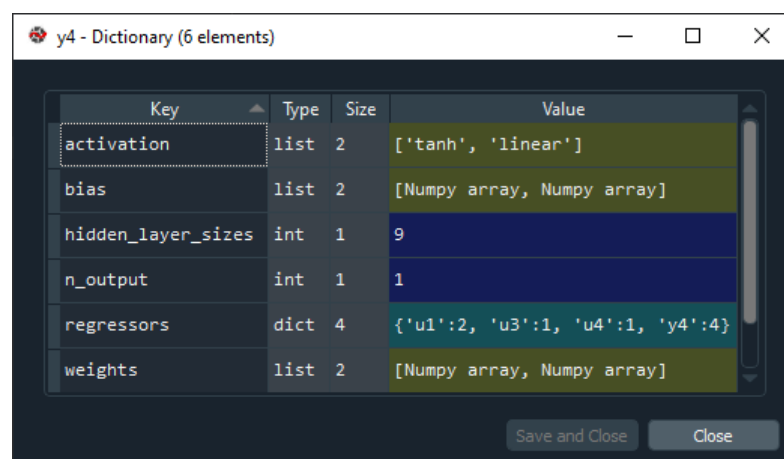
Figure 28 – An instance of the `analysis` dictionary


Key	Type	Size	Value
dependency masks	dict	12	{'y1': 'N/A', 'y2': 'N/A', 'y3': 'N/A', 'y4': 'N/A', 'y5': 'N/A', 'y6': 'N/A ...
plot evaluation	dict	12	{'y1': 'Horizon: 50, model SSE: 4.09817, baseline: 6.77811', 'y2': 'Hori ...
selected regressors	dict	12	{'y1': ['u1 = 3', 'u2 = 1', 'u4 = 1', 'y1 = 3'], 'y2': ['u1 = 1', 'u4 = ...
time info	dict	18	{'y1': 'input selection: 88.14 mins, K selection: 17.29 mins', 'Max K s ...
u independent models	list	0	[]
u participation	dict	4	{'u1': ['y1', 'y2', 'y3', 'y4', 'y5', ...], 'u2': ['y1', 'y3', 'y4', 'y5 ...

Source: by the author

## 5.7 Model exportation and plotting

For exporting the obtained models to the MPA, the `model dictionary` is created. Figure 29 shows an example of an entry for the model respective to output  $y_4$ . There is information about the used activation functions, bias, weights, number of nodes in each layer and the selected value for the `regressors`.

Figure 29 – An entry of the `model dictionary`


Key	Type	Size	Value
activation	list	2	['tanh', 'linear']
bias	list	2	[Numpy array, Numpy array]
hidden_layer_sizes	int	1	9
n_output	int	1	1
regressors	dict	4	{'u1': 2, 'u3': 1, 'u4': 1, 'y4': 4}
weights	list	2	[Numpy array, Numpy array]

Source: by the author

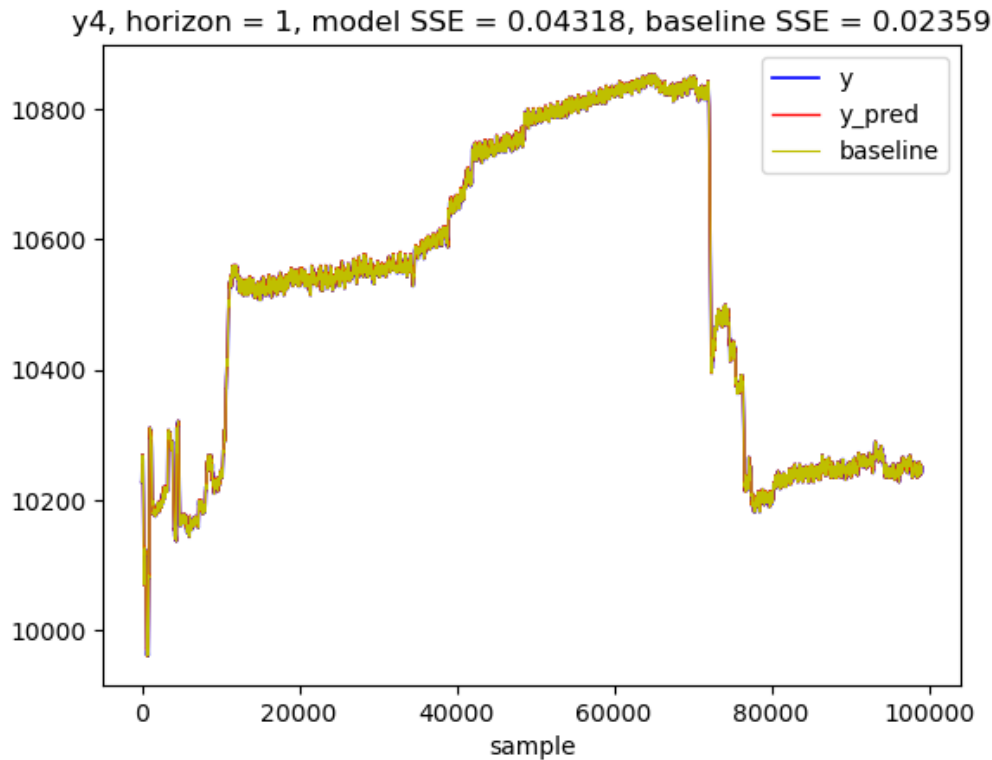
These parameters are organized in a pre-agreed format. The `model dictionary` object, which contains the entries of all created models, is saved as a `.pickle` file. An external program, previously developed by my supervisor, is responsible for reading and converting the Python dictionary into an XML, which can then be read by the MPA.

The prediction of the created models, for the whole dataset, can be plotted via the `single_plot` and `multiplots` functions. The `single_plots` function creates a plot for each one of the created models. The plots include the predicted values, the real process values and the baseline values.

The baseline values, as mentioned in section 3.3.4, are obtained by repeating the initial value for the output trough the whole horizon window. This is done for each example window that was created with the `recursive_sets` function.

Figure 30 shows an example of a single plot. The title is composed by the output for which the plot was made, the prediction horizon that was employed and the calculations of the SSE loss values for the model predictions and for the baseline.

Figure 30 – Single plot example



Source: by the author

The multiplots are plots which contain many single plots in one figure. A parameter in `exec_cfg` can be configured to determine the multiplots dimensions. In the multiplots, the baselines are not plotted to not overpopulate the figures. Examples of single plots and multiplots can be seen in the next chapter, where the the obtained results are presented and discussed.

## 6 Results

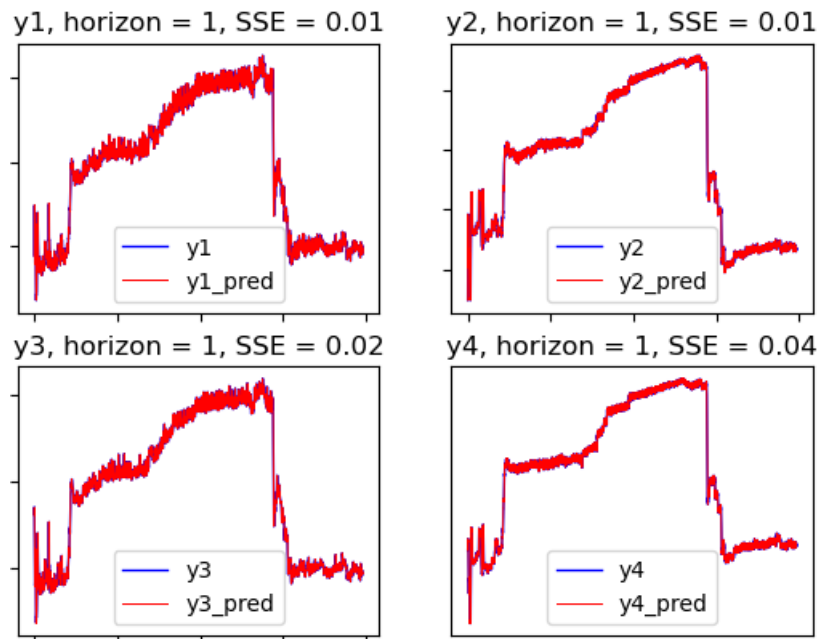
This chapter contains the analysis and discussion of the obtained results. The sections are presented according to the chronological order of the events, so that the decision path taken during the project can be followed.

The dataset used for obtaining the neural network models contains 99.1k samples from the gas compression system during normal operation conditions. There are five manipulated variables  $u$  and twelve output variables  $y$ . Following the MISO approach, a model is created for each one of the outputs, and the MIMO system can be represented by the combination of these models.

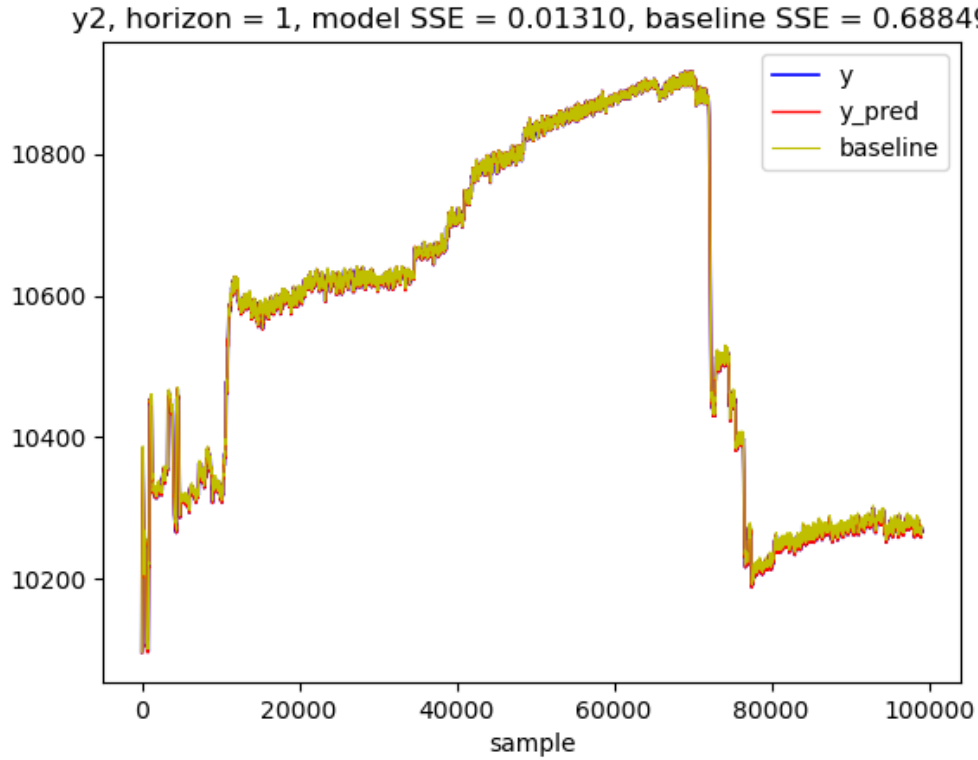
### 6.1 The first attempt

In the first attempt, the serial-parallel training configuration was employed, as mentioned and justified in section 4.2, which is equivalent to using a prediction horizon equal to one. The effect of using this configuration, as discovered later, is that the models are trained for predicting only one step ahead. A model for each one of the twelve outputs was created. Figure 31 shows the multiplot for the first four output models, and Figure 32 shows the single plot for the model respective to output  $y_2$ .

Figure 31 – Multiplot for the first four models, trained in serial-parallel



Source: by the author

Figure 32 – One step ahead predictions for  $y_2$  model, trained in serial-parallel

Source: by the author

Upon first glance, the responses look very decent, but when inspecting the obtained **analysis dictionary** (Figure 33), in particular the entry “ $u$  independent models”, which lists which of the created models have no manipulated variables selected as inputs, it can be concluded that the input selection algorithm relied almost exclusively on the previous values of the outputs when selecting the inputs for the models.

Figure 33 – **analysis dictionary** obtained in the first attempt

analysis\_dict - Dictionary (6 elements)

Key	Type	Size	Value
dependency masks	dict	12	{'y1': 'N/A', 'y2': 'N/A', 'y3': 'N/A', 'y4': 'N/A', 'y5': 'N/A', 'y6': 'N/A' ...}
plot evaluation	dict	12	{'y1': 'Horizon: 1, model SSE: 0.00877, baseline: 0.02702', 'y2': 'Horiz ...}
selected regressors	dict	12	{'y1': ['u1 = 0', 'u2 = 0', 'u3 = 1', 'u4 = 0', 'u5 = 0', ...], 'y2': [' ...}
time info	dict	18	{'y1': 'input selection: 132.85 mins, K selection: 111.73 mins', 'Max K ...}
u independent models	list	7	['y2', 'y4', 'y5', 'y6', 'y7', 'y9', 'y12']
u participation	dict	5	{'u3': ['y1'], 'u4': ['y3'], 'u5': ['y8'], 'u1': ['y10'], 'u2': ['y10', 'y1' ...}

Source: by the author

After discussing this problem with my supervisor, we came to the conclusion that the training configuration should be changed to parallel. With the proposed alteration, the models will be created and trained having to make predictions within a prediction horizon, which will likely force them to prioritize the identification of the relations between the output and the manipulated variables (i.e. to capture the system behavior), rather than simply relying on the replication of the previous output values provided by the dataset.

Changing the training configuration was one of the activities that required the greatest investment of time. It was necessary to study how to create custom models using `tf.keras`, as guided by [17], so that the output of a model can be fed back as an input for the next timestep.

This change also encouraged and made possible the testing of different types of recursive neural networks structures, namely the LSTM, GRU and the Standard RNN. In consequence, a dual layer LSTM was adopted for use in the input selection algorithm. It was verified that using this structure grants more consistent input selection results than the DLP, while also prevents the occurring of vanishing/exploding gradient problems, which in turn allows the use of higher prediction horizons and discourages the network from relying on simple output replication.

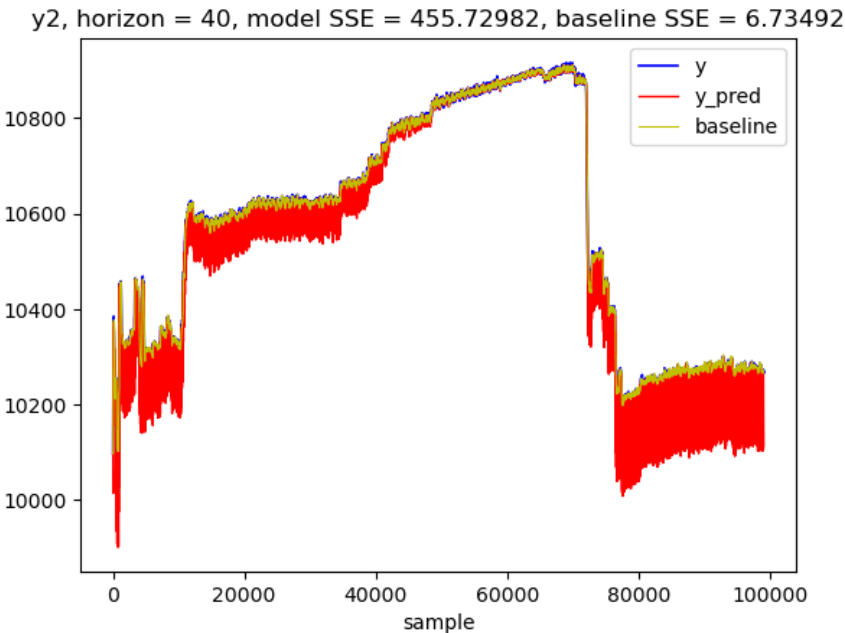
In the K selection algorithm, a custom version of the DLP structure is used, which allows training to be performed in the parallel configuration. The LSTM structure was not used here as well because the current MPC application implemented in the MPA doesn't offer support for internal state recursion.

This endeavor culminated in the creation of the `custom_models.py` module, which contains the implemented model classes. It was only then that it was necessary to create the `recursive_sets` function, located in `data_utils.py`, which is responsible for organizing the X and Y sets into three dimensions (timestep, features and example), which is the required input format for training recursive neural networks using `tf.keras`.

After the development of the new features, the obtained models could be trained and used for making predictions within an horizon greater than one. Figure 34 shows the same model from Figure 32 having to make predictions using a prediction horizon of 40 steps. It can be noticed that the model had failed to capture any behavior of the system during training. This happened to all models created in the first attempt, as depicted in Figure 35, all of them have performed considerable worse than the respective baselines.

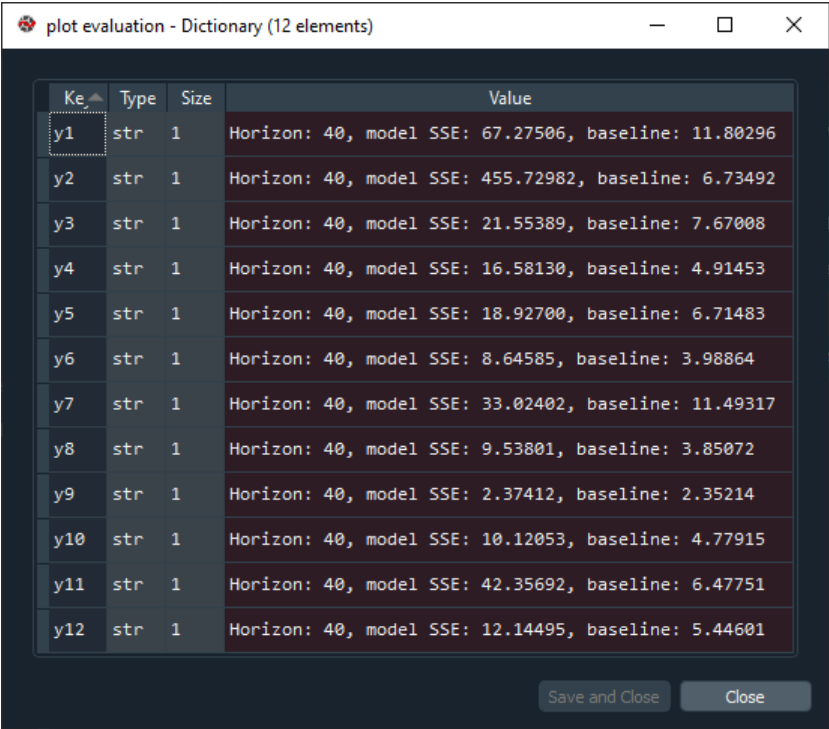
It was concluded that poor results were obtained because the created models were only good at predicting one step ahead, while, for being used by the MPC, it is required of them to make good predictions over a greater range. The output of  $y_2$  model was chosen to be shown in the figures because its performance discrepancy, when changing the prediction horizon, is very evident on the plots.

Figure 34 – 40 step ahead predictions for  $y_2$  model, trained serial-parallel



Source: by the author

Figure 35 – 40 step ahead evaluation of the models trained in serial-parallel



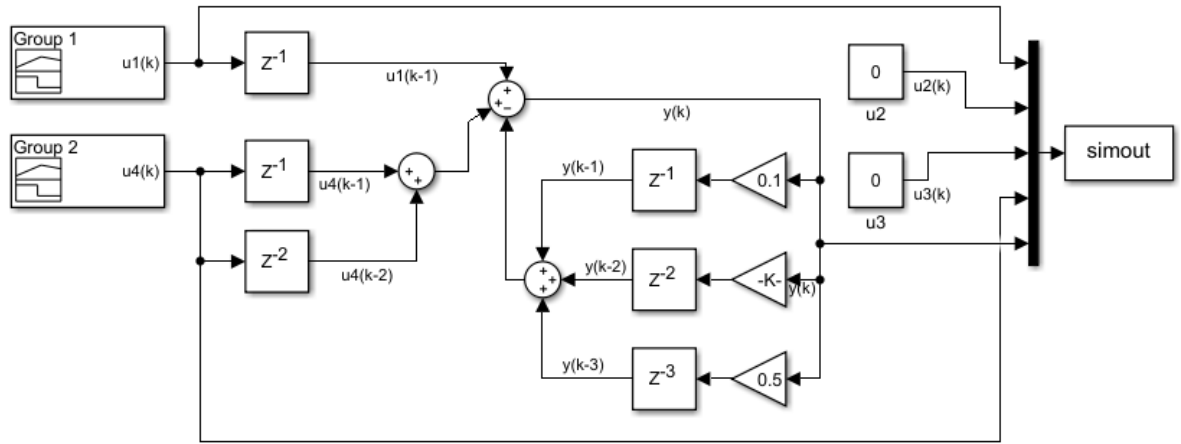
Source: by the author

## 6.2 The Simulink validation model

Before starting the next attempt at re-obtaining the models with the real process dataset, my supervisor suggested the experiment of a dataset generated from a known model, with the objective of validating the input selection algorithm, which was not able to detect the relations between the process outputs and the manipulated variables in the first attempt with the real process dataset.

A model for this purpose was created on Simulink, which is a graphical block diagramming tool from MATLAB, a widely known software for numerical computing. This model is referred to as the validation model. Figure 36 shows the implemented block diagram model for generating the dataset.

Figure 36 – The validation model



Source: by the author

When doing an experiment for collecting data for the purpose of system identification, it is necessary to apply a wide range of different input signals, in order to excite the system in the most different ways possible, so that the generated dataset contains internally the dynamics of the process.

The applied input signals for the manipulated variables  $u_1$  and  $u_4$  were a composition of steps, sinusoids, ramps and small noises. The signals for  $u_3$  and  $u_4$  were kept at zero, but their measurements were included in the dataset. The reason for that is to check if the algorithm would have the ability to ignore variables that do not have influence on the output.

The output of the block diagram model can be represented by the NARX function:

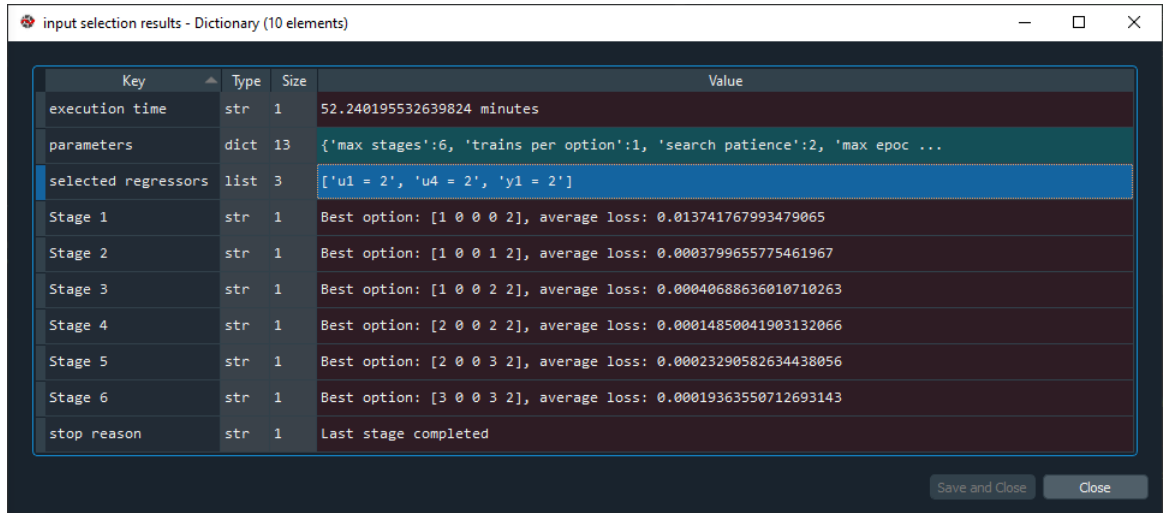
$$y_1 = f(u_1(k-1), u_4(k-1), u_4(k-2), y_1(k-1), y_1(k-2), y_1(k-3))$$



The value for **regressors** which correctly represents the model is  $[\text{order}(u_1) = 1, \text{order}(u_4) = 2, \text{order}(y_1) = 3]$ , this is the expected value for **regressors** that should be returned by the input selection algorithm.

The generated dataset has 10k samples. The first test with it was in the serial-parallel configuration (horizon=1), and the input selection algorithm failed to select good inputs. However, in the next test, the parallel configuration was used, and the algorithm managed to select almost the same **regressors** that was used for generating the dataset, as can be seen in Figure 37. The only mistake was considering an additional order for the input  $u_1$ , which in turn is not so harmful for the performance.

Figure 37 – Input selection results for the validation model



Key	Type	Size	Value
execution time	str	1	52.240195532639824 minutes
parameters	dict	13	{'max stages':6, 'trains per option':1, 'search patience':2, 'max epoc ...
selected regressors	list	3	['u1 = 2', 'u4 = 2', 'y1 = 2']
Stage 1	str	1	Best option: [1 0 0 0 2], average loss: 0.013741767993479065
Stage 2	str	1	Best option: [1 0 0 1 2], average loss: 0.0003799655775461967
Stage 3	str	1	Best option: [1 0 0 2 2], average loss: 0.00040688636010710263
Stage 4	str	1	Best option: [2 0 0 2 2], average loss: 0.00014850041903132066
Stage 5	str	1	Best option: [2 0 0 3 2], average loss: 0.00023290582634438056
Stage 6	str	1	Best option: [3 0 0 3 2], average loss: 0.00019363550712693143
stop reason	str	1	Last stage completed

Source: by the author

It should be noted that, in order to force the algorithm to stop favouring the replication of the previous output value, a high value for the prediction horizon had to be set. In this case, an horizon of 150 was used, as can be seen in Figure 38. Using a prediction horizon higher than that may not be advisable, due to the possible occurrence of vanishing/exploding gradient problems, even with the use of the LSTM structure.

With almost the correct **regressors** selected, the K selection algorithm was performed, resulting in the selection of  $K = 7$  for the definitive DLP model. The plot of the response of the validation model is shown in Figure 39. It can be noticed that the achieved performance is considerably better than the baseline.

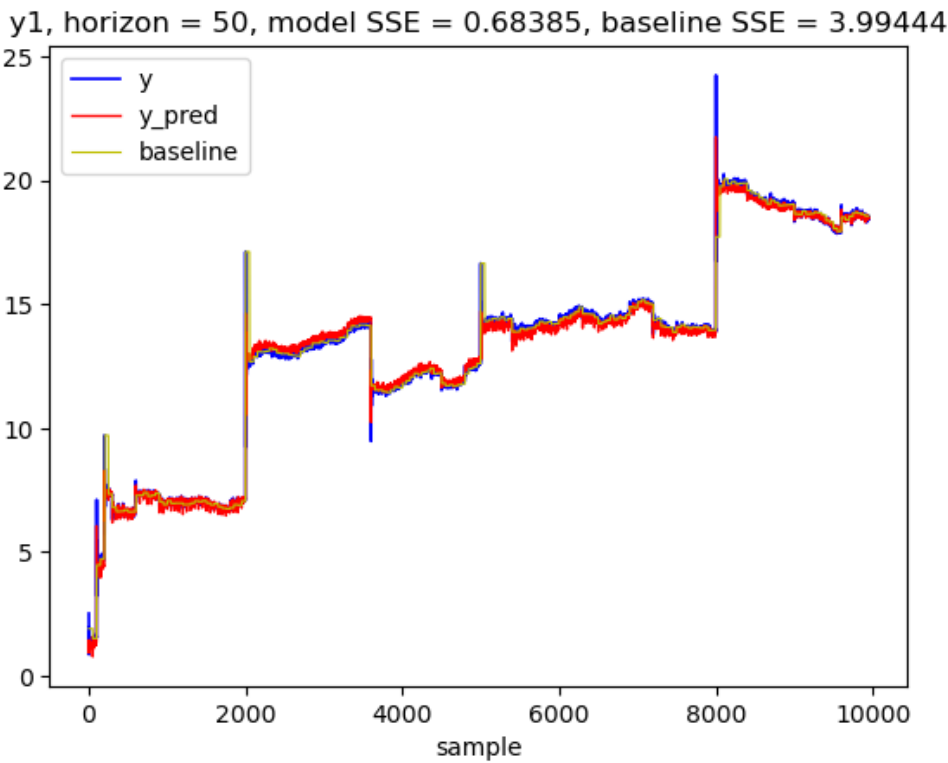
Figure 40 shows a close inspection of the obtained response. It can be noticed that the model successively replicates the system behavior, although with a small gain error. This test with the validation model provided evidence that supports the effectiveness of the implemented algorithms, at least when a representative dataset is provided.

Figure 38 – Parameters for the input selection of the validation model

Key	Type	Size	Value	Key	Type	Size	Value
acceptable loss	bool	1	False	resampling factor	int	1	1
early stop patience	int	1	3	search patience	int	1	2
hidden layer nodes	int	1	8	starting y order	int	1	2
horizon	int	1	150	target loss	bool	1	False
max epochs	int	1	1000	trains per option	int	1	1
max stages	int	1	6	validation size	int	1	0
min delta loss	bool	1	False				

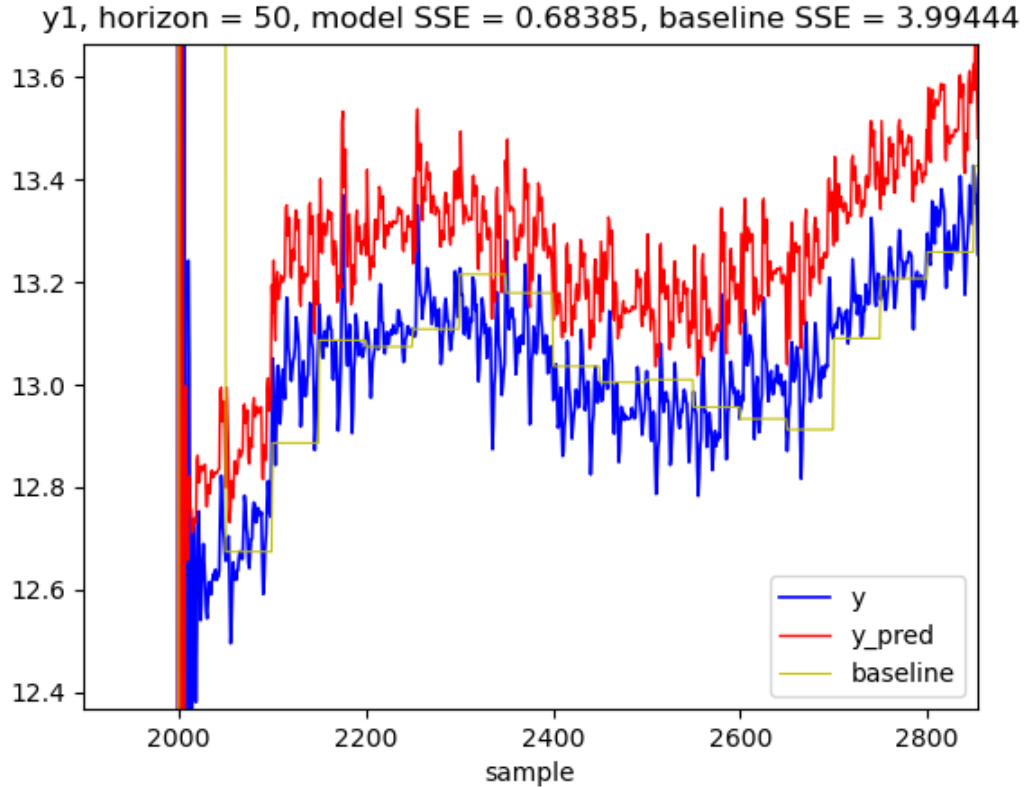
Source: by the author

Figure 39 – Plot of the validation model predictions



Source: by the author

Figure 40 – Zoomed plot of the validation model predictions



Source: by the author

### 6.3 The second attempt

The models for the real process dataset were re-obtained, now using the parallel training configuration, the LSTM structure for the input selection algorithm models and the custom recursive DLP for the K selection algorithm.

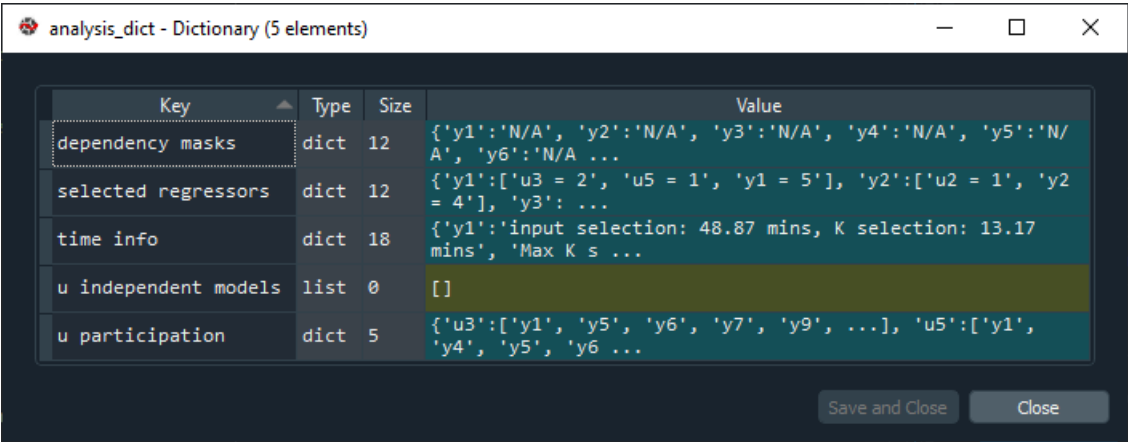
Figure 41 shows that the obtained models now have manipulated variables being considered as inputs. However, the performance were, still, considerably worse than the baseline, even at high prediction horizons, which implies that the models had, once again, failed to capture the system dynamics.

Another dataset from the real process operating in a different day was tested, but the data was very similar, so the achieved results were also poor. Multiple prediction horizon values were tested. It was also tried to skip the `trim_data` call before running the input selection algorithm, in order to force the analysis of all process variables as candidate inputs, but the obtained models were also unsuccessful.

Figure 42 illustrate the response of *y2* model trained in the parallel configuration. Comparing it to the response obtained by the same model trained in the serial-parallel configuration (Figure 34), it can be noticed an improvement in the performance, although

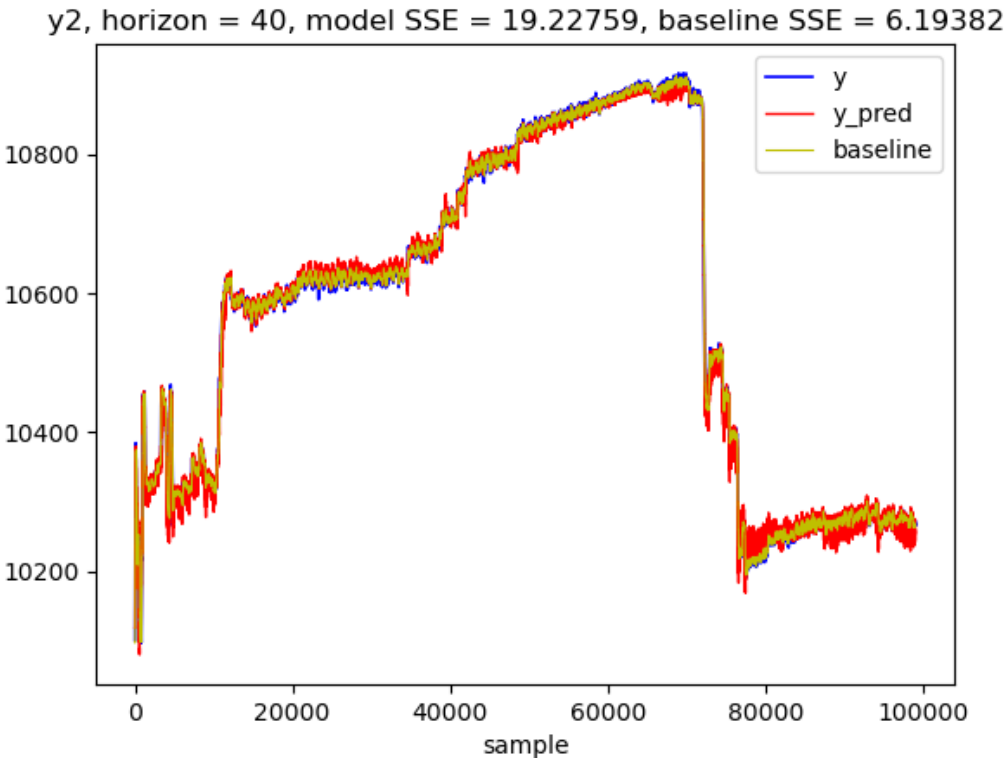
this has not much relevance, because it is still worse than the baseline.

Figure 41 – Analysis dictionary from the second attempt



Source: by the author

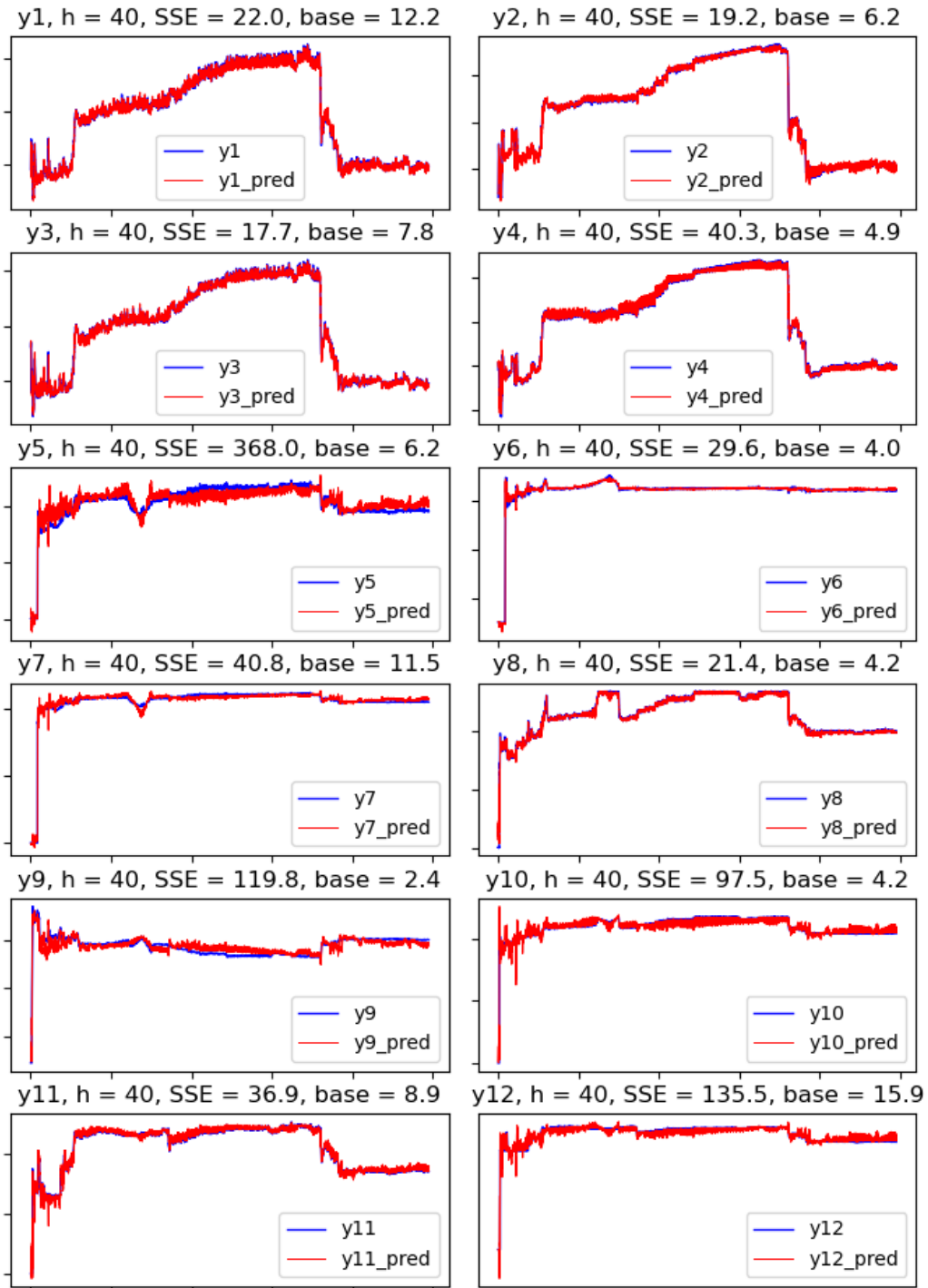
Figure 42 – Plot of the  $y_2$  model predictions



Source: by the author

Figure 43 shows the multiplot for all the twelve models obtained in the second attempt, trained in the parallel configuration using a prediction horizon equal to 40. No model could surpass the baseline performance.

Figure 43 – Multiplot of the models obtained in the second attempt



Source: by the author

Me and my supervisor discussed the probable reasons of why the neural networks were failing to capture the system dynamics, even after changing the training configuration and using the LSTM for the input selection algorithm. We raised to possibility that the used dataset was not ideal for the system identification procedure, and did not contained

relevant information about the process dynamics and the relationship between the output and the manipulated variables.

The dataset samples were acquired with the process during routine operation, when, most of the time, the outputs are on steady-state, with rare changes made to setpoints. This condition makes the over time accumulated measurement noises to hinder the identification process.

This problem suggest the obtainment of a new dataset from the gas compression system, exclusively for the purpose of serving system identification procedures. A multiple range of different signals would have to be applied on each of the manipulated variables, in order to excite the system in as many ways as possible, while keeping it within safety thresholds.

However, such complex experiment may not be possible to be performed with the real system, nor may be worth the effort, when taking into account the financial costs of having to interrupt the production for doing so. Therefore, the use of a simulation model may be the only viable solution. In the next section, the algorithm is tested with a dataset generated from a simulation of the gas compression system.

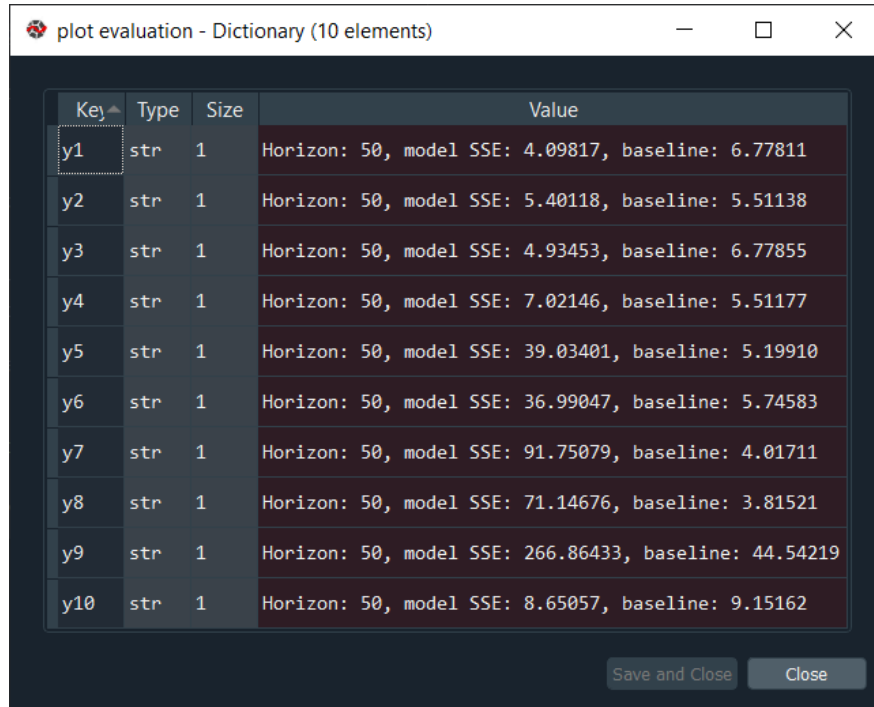
## 6.4 The simulation dataset

In another attempt to validate the implemented algorithms, a dataset from a simulation of the gas compression system was used. This dataset was generated using the EMSO simulator, in a previous iteration of the project [14]. The EMSO simulator is an open-source alternative to Simulink.

It should be noted that the simulation system bears little resemblance to the original system, having different components and different process variables. This means that the eventually obtained models cannot be used for representing the real system. The dataset contains 18k samples, composed of four manipulated variables  $u$  and thirty-six output variables  $y$ .

During the system identification procedure, it was preferred not to create neural models for all the thirty-six outputs. Instead, it was created models for only the first ten outputs, since the algorithm's performance on them would likely be similar for the rest. Also, the parallel training configuration was used. Four out of the ten obtained models managed to have better performance than baseline, as depicted in Figure 44, which shows the `plot evaluations` entry from the `analysis dictionary`.

Figure 44 – Plot evaluations for the models obtained from the simulation dataset



Key	Type	Size	Value
y1	str	1	Horizon: 50, model SSE: 4.09817, baseline: 6.77811
y2	str	1	Horizon: 50, model SSE: 5.40118, baseline: 5.51138
y3	str	1	Horizon: 50, model SSE: 4.93453, baseline: 6.77855
y4	str	1	Horizon: 50, model SSE: 7.02146, baseline: 5.51177
y5	str	1	Horizon: 50, model SSE: 39.03401, baseline: 5.19910
y6	str	1	Horizon: 50, model SSE: 36.99047, baseline: 5.74583
y7	str	1	Horizon: 50, model SSE: 91.75079, baseline: 4.01711
y8	str	1	Horizon: 50, model SSE: 71.14676, baseline: 3.81521
y9	str	1	Horizon: 50, model SSE: 266.86433, baseline: 44.54219
y10	str	1	Horizon: 50, model SSE: 8.65057, baseline: 9.15162

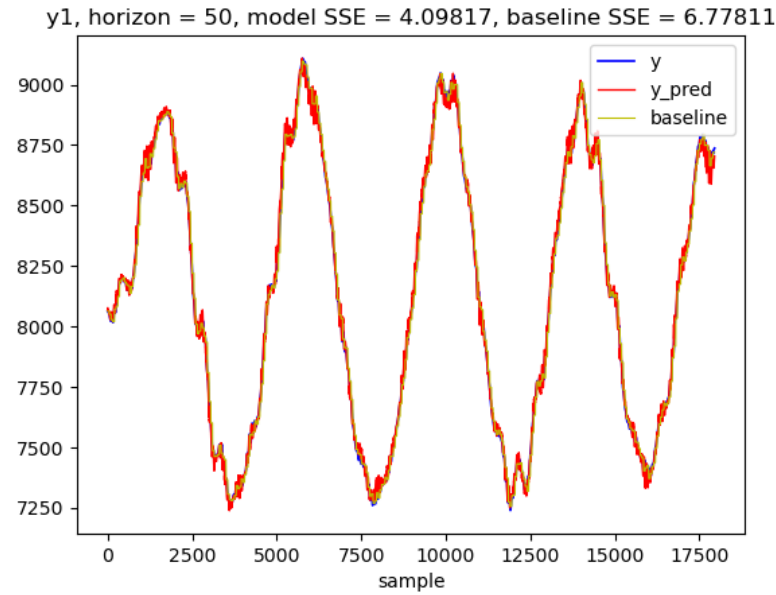
Source: by the author

It has to be noted that, for the models which performed worse than baseline, excluding the  $y_4$  model, their performances were considerably poor. This is probably due to these outputs having no internal relation to the  $u$  variables of the simulation system, which we had no access to verify. Skipping the `trim_data` function call before running the input selection could solve the problem, but then the procedure would take a long time, because all forty variables would be considered potential inputs.

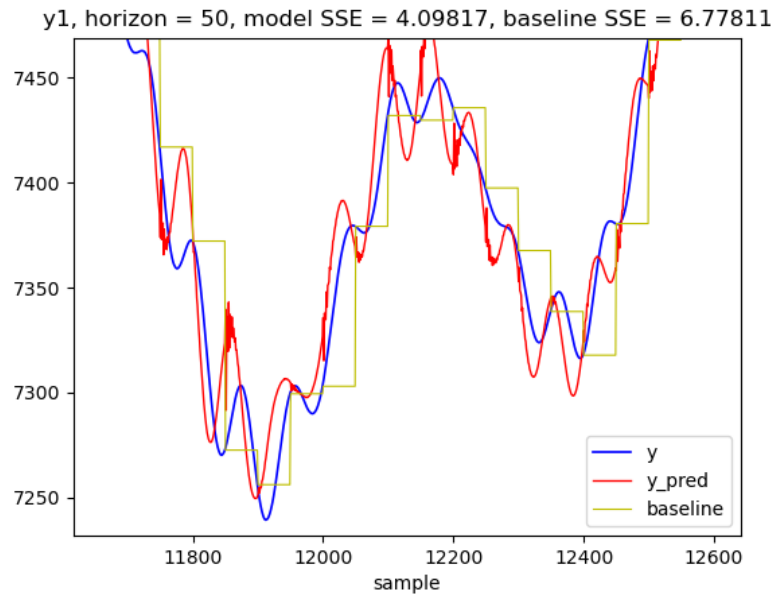
However, some of the obtained models had good performance, which did not happened for any of the twelve models obtained with the dataset from the real gas compression process, when using a prediction horizon equal to 40 for training.

Figure 45 shows the response of the  $y_1$  model, while Figure 46 shows it more closely. It can be seen that the model is replicating the system behavior, although with some gain discrepancies. If this model were to be applied in a real control system, an algorithm like the PNMPC should be robust enough to correct the modeling errors.

Figure 47 shows the multiplot for the predictions of the ten obtained models. It can be noted that the outputs have a sinusoidal shape, probably due to the application of only sinusoidal signals when the process was simulated. This may have hampered the identification procedure. Unfortunately, nor the software or the modeled system file were available, so a new dataset couldn't be generated.

Figure 45 – Plot of the  $y_1$  model from the EMSO dataset

Source: by the author

Figure 46 – Zoomed plot of the  $y_1$  model from the EMSO dataset

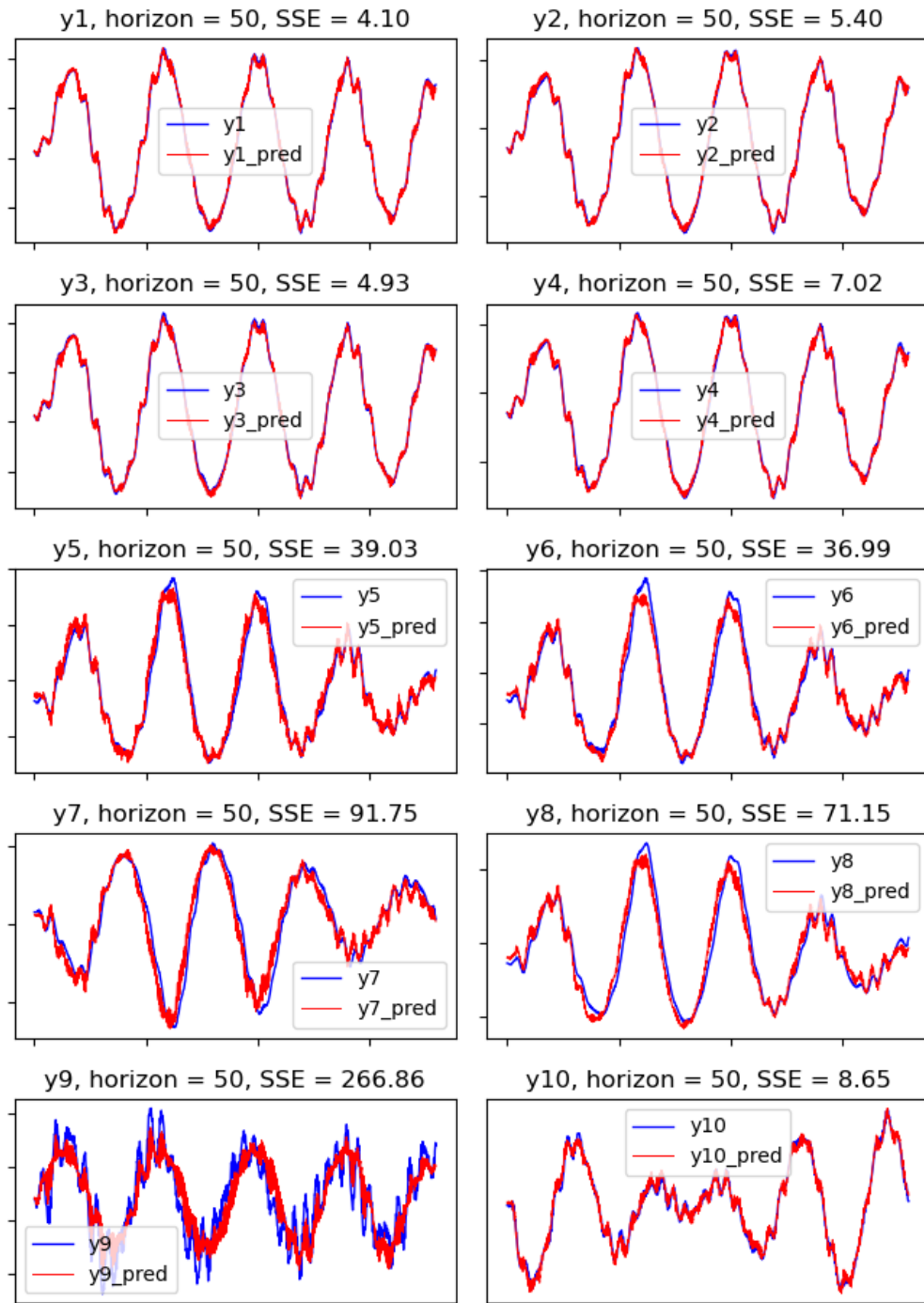
Source: by the author

In addition to the results achieved with the validation system, this test with the simulation dataset provides further evidence that supports the effectiveness of the implemented algorithms. In the concluding chapter, some possible ways to solve the



problems obtained with the real process dataset are discussed, along with some ideas of what can be done in future implementations.

Figure 47 – Multiplot of the first four models from the simulation model



Source: by the author

## 7 Conclusion

In this project, it was created a system identification program that receives a dataset and obtains neural network models that represent the process. The models were aimed to be used by the PNMPC as prediction models of the gas compression system, in specific for the purpose of calculating the free response component.

For the implementation, it was necessary to investigate the use of neural networks as process models, the possible structures available and how to implement, train and evaluate neural networks using the Python language. It was also necessary to conduct a study on system identification procedures, including the possible training configurations and the investigation of the literature to see what has been done in practice.

Details about the software implementation, including the employed technologies and the general explanation of the functioning of the program were presented. Then, the obtained results, for all the datasets tested, were analyzed and discussed. It was verified the inability of the algorithm for obtaining, from the currently available dataset of the real process, models that could satisfactorily represent the gas compression system.

However, good results were obtained with the validation and simulator model datasets, which suggest the conception that a new dataset representing the real process must be generated. In the ideal case, this would be obtained via a system identification experiment procedure, where the plant would be excited by a multiple range of different input signals. This would facilitate the projection of process dynamics in the measured outputs, resulting in a much more meaningful dataset. Nevertheless, an experiment like this might be infeasible to do in practice.

A more realistic solution would be to create a new simulation model, having the same variables and components as the real process. Another possible path is to create a dataset composed of the real system measurements only during transient conditions, eventually generated by changes in the setpoints.

### 7.1 Future developments

For future developments, there is an interesting idea that could be investigated, which is the possibility of performing the online linearisation of the obtained neural models. Then, while the nonlinear neural model is used for calculating the free response, the linearised version is used for the forced response. In that case, a performance comparison should be made between the linearised neural models obtained with this technique and the linearised models which are currently obtained with the step-response linearisation

technique employed by the PNMPC.

It could also be investigated the use of neural network based suboptimal MPC algorithms. The book that served as the main source of study for this project [2] contains many chapters about the implementation of these algorithms, with some of them being capable of having comparable performance to conventional nonlinear optimisation algorithms, while demanding much less computational power.

# References

- 1 BERENGUEL, M.; ARAHAL, M. R.; CAMACHO, E. F. Modelling the free response of a solar plant for predictive control. *Control Engineering Practice*, n. 6, p. 1257—1266, 1998. Cited 3 times in pages 5, 33, and 34.
- 2 LAWRYNCZUK, M. *Computationally Efficient Model Predictive Control Algorithms: A neural network approach*. Warsaw, Poland: Springer, 2014. Cited 16 times in pages 5, 13, 14, 18, 19, 20, 21, 22, 27, 31, 34, 35, 36, 37, 46, and 66.
- 3 CAMACHO, E. F.; BORDONS, C. Nonlinear model predictive control: An introductory review. *Assessment and Future Directions of Nonlinear Model Predictive Control*, p. 1–16, 2007. Cited in page 10.
- 4 VETTORAZZO, C. M. Model predictive control of gas compression station in off-shore production platforms. 2016. Cited 3 times in pages 10, 11, and 13.
- 5 NORMEY-RICO, J. E. et al. Relatório técnico 2. *Desenvolvimento de Algoritmos de Controle Preditivo Não-Linear e de Avaliação de Desempenho de Controladores Preditivos para Plataformas de Produção de Petróleo (Fase 2)*, 2019. Cited in page 11.
- 6 AGUIAR, M. A.; CODAS, A.; CAMPONOGARA, E. Systemwide optimal control of offshore oil production networks with time dependent constraints. v. 48, n. 6, p. 200–207, 2015. Cited in page 12.
- 7 OIL & Gas Process Engineering. 2020. Disponível em: <<http://www.oilngasprocess.com/gas-production-facility/centrifugal-compressors-surge-control-and-stonewalling.html>>. Acesso em: 01 dec. 2020. Cited in page 13.
- 8 MORARI, M.; LEE, J. Model predictive control: past, present and future. *Computers and Chemical Engineering*, n. 23, p. 667–682, 1999. Cited in page 13.
- 9 PLUCENIO, A. Desenvolvimento de tecnicas de controle nao linear para elevacao de fluidos multifasicos. thesis (phd). PPGEAS, DAS, Universidade Federal de Santa Catarina, Florianopolis-SC-Brasil, 2010. Cited in page 15.
- 10 HUSSAIN, M. Review of the applications of neural networks in chemical process control—simulation and online implementation. *Artificial Intelligence in Engineering*, n. 13, p. 55–68, 1999. Cited in page 18.
- 11 HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, n. 2, p. 359–366, 1989. Cited in page 18.
- 12 NG, A. Y. *Deep Learning Specialization*. Coursera, 2020. Disponível em: <<https://www.coursera.org/specializations/deep-learning>>. Acesso em: 20 nov. 2020. Cited 3 times in pages 20, 21, and 25.
- 13 HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, München, Germany, n. 9, 1997. Cited in page 20.

- 14 MARCON, R. F. Uso de redes neurais para obtenção de modelos de predição para controladores preditivos lineares. *DAS-5511: Projeto de Fim de Curso*, 2019. Cited 3 times in pages 23, 37, and 61.
- 15 ARAHAL, M. R.; BERENGUEL, M.; CAMACHO, E. F. Neural identification applied to predictive control of a solar plant. *Control Engineering Practice*, n. 6, p. 333—344, 1998. Cited 4 times in pages 27, 28, 32, and 42.
- 16 PHAM, D.; LIU, X. Neural networks for identification, prediction and control. In: \_\_\_\_\_. Cardiff, UK: Springer, 1995. cap. 2, p. 27–29. Cited 3 times in pages 29, 30, and 31.
- 17 TENSORFLOW. *Time series forecasting*. 2020. Disponível em: <[https://www.tensorflow.org/tutorials/structured\\_data/time\\_series](https://www.tensorflow.org/tutorials/structured_data/time_series)>. Acesso em: 20 nov. 2020. Cited 3 times in pages 31, 39, and 53.
- 18 KORBICZ, J.; KOŚCIELNY, J. M. Modeling, diagnostics and process control: Implementation in the diaster system. In: \_\_\_\_\_. Poland: Springer, 2011. cap. 3, p. 101. Cited 2 times in pages 31 and 34.
- 19 BOMBERGER, J.; SEBORG, D. Determination of model order for narx models directly from input-output data. *J. Proc. Cont.*, v. 8, n. 5, 6, p. 459–468, 1998. Cited in page 33.
- 20 CARVALHO, C. B.; CARVALHO, E. P.; RAVAGNANI, M. A. S. S. Combined neural networks and predictive control for heat exchanger networks operation. *Chem. Ind. Chem. Eng. Q.*, n. 26, p. 125–134, 2020. Cited in page 33.
- 21 ARAHAL, M. R. Identificación y control mediante redes de neuronas. PhD thesis. Universidad de Sevilla, Spain, 1996. Cited in page 33.
- 22 BILLINGS, S. A. *Nonlinear System Identification: Narmax methods in the time, frequency, and spatio-temporal domains*. United Kingdom: Willey, 2013. Cited in page 37.
- 23 de CARVALHO, C. *neural-nets-mpc*. 2020. Disponível em: <<https://github.com/christopherdec/neural-nets-mpc>>. Acesso em: 20 nov. 2020. Cited 2 times in pages 39 and 45.