

# Input convex neural networks in nonlinear predictive control: A multi-model approach

Maciej Ławryńczuk<sup>a</sup>

<sup>a</sup> Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Control and Computation Engineering, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland

## ARTICLE INFO

### Article history:

Received 20 January 2022

Revised 30 July 2022

Accepted 15 September 2022

Available online 27 September 2022

Communicated by Zidong Wang

### Keywords:

Input convex neural networks

Multi-models

Model Predictive Control

Polymerisation reactor

## ABSTRACT

The presented input convex neural multi-modelling approach to Model Predictive Control (MPC) has two essential advantages. Firstly, the MPC algorithm solves only convex optimisation tasks; complex not convex and multi-modal problems are not considered. Secondly, the model is trained in a simple one-step-ahead configuration and it is not used recurrently for prediction. Two neural multi-model structures are considered: fully input convex and partially input convex ones. The fully convex model structure turns out to be too restrictive for the considered polymerisation reactor benchmark process. The partially convex model leads to very good prediction accuracy and excellent control quality. Development of fully and partially input convex multi-models is thoroughly described, particularly selection of the number of the hidden nodes. Additionally, low effectiveness of a linear multi-model is discussed for comparison.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction, motivation and contribution

### 1.1. Research area and the main problem

Model Predictive Control (MPC) is an advanced control approach that has two special features: a) the use of a dynamical model of the controlled process for online prediction, b) the use of an optimisation procedure to find online the best control policy [28,41]. Due to good control quality, MPC algorithms find many applications. Typical industrial applications are: chemical reactors [26], electromagnetic mills [34], wastewater treatment plants [33], hydroelectric generating units [47] and activated sludge processes [46]. Nowadays, MPC methods are increasingly popular in fast embedded systems; example applications are: autonomous driving systems [27], unmanned surface [35] and aerial vehicles [38].

When a nonlinear dynamical model is used for prediction in MPC, a general nonlinear optimisation problem must be solved at each sampling instant online. Two main disadvantages of such a rudimentary nonlinear MPC approach are: significant difficulty which may require long calculation time and the minimised cost function may be multi-modal and non-convex, meaning that shallow local minima may be found rather than a global solution. Several methods have been formulated to reduce computational difficulty of nonlinear MPC:

1. Successive online linearisation may be used, which leads to quadratic optimisation tasks which have only one global solution [21,25,31]. Nonlinear optimisation is not necessary.
2. In the case of block-oriented dynamical models comprised of independent linear dynamic and nonlinear static blocks, an inverse static model may be used to compensate for process nonlinearity [25,37]. It also leads to quadratic optimisation.
3. When the Koopman operator lifts the nonlinear dynamics into a higher-dimensional space, a linear model may describe a nonlinear process. As a result, a quadratic MPC optimisation task is obtained [17].
4. Neural networks may approximate the explicit MPC control law or time-varying coefficients of the control law [3,22,30].
5. Constrained explicit nonlinear MPC algorithms utilise a set of unconstrained, simple local control laws [12,14]. No online optimisation is used, but, unfortunately, the necessary number of such controllers may be high.
6. Convexification of the multi-modal MPC cost function is possible [10,13,36]. The resulting cost function is not expected to be quadratic, but convexity guarantees that the global solution is easily found.
7. Globally optimal nonlinear MPC based on multi-parametric disaggregation may be used [43]. In this approach, the nonlinear optimisation task is repeatedly relaxed online into a mixed-integer linear programming problem which generates a starting point for a nonlinear solver.

E-mail address: [Maciej.Lawrynczuk@pw.edu.pl](mailto:Maciej.Lawrynczuk@pw.edu.pl)

## Nomenclature

### Model Predictive Control

|                                    |                                                                                                                                |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| $J(k)$                             | the MPC cost-function minimised at each sampling instant                                                                       |
| $N, N_u$                           | prediction and control horizons                                                                                                |
| $n_A, n_B$                         | integer numbers which define the order of model dynamics                                                                       |
| $u(k+p k)$                         | the future value of the manipulated variable for the sampling instant $k+p$ calculated at the current time step $k$            |
| $\mathbf{u}(k)$                    | the vector of MPC decision variables (the future manipulated variables over the control horizon)                               |
| $u^{\min}, u^{\max}$               | minimal and maximal values of the manipulated variable                                                                         |
| $y^{\min}, y^{\max}$               | minimal and maximal values of the controlled variable                                                                          |
| $y^{\text{sp}}(k+p k)$             | the future value of the set-point of the controlled variable for the sampling instant $k+p$ known at the current time step $k$ |
| $\hat{y}(k+p k)$                   | the predicted value of the controlled variable for the sampling instant $k+p$ calculated at the current time step $k$          |
| $\Delta u(k+p k)$                  | the future increment of the manipulated variable for the sampling instant $k+p$ calculated at the current time step $k$        |
| $\Delta u^{\min}, \Delta u^{\max}$ | minimal and maximal changes of the manipulated variable                                                                        |
| $\lambda$                          | the penalty coefficient in MPC                                                                                                 |

### General Symbols Used in Modelling and Prediction

|            |                                                                                                                                                                                                          |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $d(k)$     | the estimation of the unmeasured disturbance that acts on the process output at the current sampling instant $k$ (the classical model used recurrently)                                                  |
| $d(k+p k)$ | the estimation of the unmeasured disturbance that acts on the process output for the future sampling instant $k+p$ calculated at the current sampling instant $k$ (the multi-model not used recurrently) |
| $f$        | a nonlinear function that describe a classical nonlinear model                                                                                                                                           |
| $f_p$      | a nonlinear function that describe a nonlinear sub-model for the sampling instant $k+p$                                                                                                                  |
| $k$        | discrete time (the current sampling instant)                                                                                                                                                             |
| $n_A, n_B$ | integer numbers that define the order of model dynamics                                                                                                                                                  |

|                       |                                                                                                                                             |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| $u$                   | the process input (the manipulated variable)                                                                                                |
| $\mathbf{x}_d(k+p k)$ | the vector of model arguments used to calculate the unmeasured disturbance $d(k+p k)$                                                       |
| $\mathbf{x}_f(k+p k)$ | the vector of future manipulated variables used in prediction for the future sampling instant $k+p$ calculated at the current time step $k$ |
| $\mathbf{x}_p(k)$     | the current operating point vector of the process defined by past measurements of the process input and output signals                      |
| $y$                   | the process output (the controlled variable)                                                                                                |
| $y(k+p k)$            | the model output for the sampling instant $k+p$ calculated at the current time step $k$                                                     |

### Input Convex Neural Network

|                             |                                                |
|-----------------------------|------------------------------------------------|
| $b_i$                       | bias coefficients                              |
| $K_i$                       | the number of hidden nodes in the $i$ th layer |
| $L$                         | the number of layers                           |
| $W_i^{(z)}$ and $W_i^{(x)}$ | weights of the network                         |
| $x$                         | the network input vector                       |
| $y$                         | the network output vector                      |
| $z_i$                       | the output vector of the $i$ th layer          |

### Partially Input Convex Neural Network

|                                                           |                                                                              |
|-----------------------------------------------------------|------------------------------------------------------------------------------|
| $\tilde{b}_i$                                             | bias coefficients in the $v$ -path                                           |
| $b_i^{(z)}, b_i^{(x)}, b_i$                               | bias coefficients in the $x$ -path                                           |
| $K_i$                                                     | the number of hidden nodes in the $i$ th layer                               |
| $L$                                                       | the number of layers                                                         |
| $u_i$                                                     | the output vector of the $i$ th layer related to the $v$ -path               |
| $v$                                                       | the network input vector that does not have a convex influence on the output |
| $\tilde{W}_i$                                             | weights in the $v$ -path                                                     |
| $W_i^{(z)}, W_i^{(zu)}, W_i^{(x)}, W_i^{(xu)}, W_i^{(u)}$ | weights in the $x$ -path                                                     |
| $x$                                                       | the network input vector that has a convex influence on the output           |
| $y$                                                       | the network output vector                                                    |
| $\varphi_i$                                               | the nonlinear activation function of the $i$ th layer in the $x$ -path       |
| $\tilde{\varphi}_i$                                       | the nonlinear activation function of the $i$ th layer in the $v$ -path       |
| $z_i$                                                     | the output vector of the $i$ th layer related to the $x$ -path               |

8. Input Convex Neural Networks (ICNNs) [1] may be used for prediction. As a result, the resulting MPC optimisation task is also convex.

## 1.2. Motivation

This work is concerned with using input convex neural networks in MPC. Specifically, two types of networks are considered: fully ICNNs and partially ICNNs introduced in [1]. Both types of ICNNs are convex in one-step-ahead prediction. To make a multiple-step-ahead prediction, a procedure described in [7] may be used in the case of ICNNs models. An alternative method that requires additional constraints in the network architecture is presented in [4]. Of course, when multiple-step-ahead prediction is required, model errors are propagated and are likely to be larger than in the one-step-ahead configuration.

Because the input convex neural network is a relatively novel structure, only a limited number of applications are reported in the literature. A Hammerstein-Wiener model comprised of two

nonlinear static blocks represented by ICNNs and one linear dynamic block is discussed in [32]. The model is used in MPC of an engine airpath system. An application of ICNNs in operation optimisation and MPC of a chemical reactor is reported in [45]. The use of ICNNs in optimisation of transient control against Denial of Service (DoS) attacks on an energy storage system is considered in [40]. An optimal voltage regulation methodology using ICNNs in a distribution network is detailed in [8]. Polyconvex networks for finite deformations are proposed in [16]. Typically, in fully and partially ICNNs, simple ReLU or LeakyReLU activation functions are used. An alternative convex neural network relying on Log-Sum-Exp functions is discussed in [5].

Typically, the process behaviour is predicted over several time steps in MPC. It means that the underlying dynamical model must be used recurrently online. In such a case, the predictions depend in a complex, i.e. nonlinear, possibly multi-modal and non-convex way on the calculated decision variables, i.e. the optimised control policy of MPC. Furthermore, recurrence may lead to inaccuracies of predictions (the prediction errors are accumulated), in

particular at the end of the horizon, as demonstrated in [20,21]. There are two modelling approaches that eliminate recurrence: the multi-model [21] and the factorised structure [2,18,39]. Both approaches allow to calculate the process predictions without any recurrence. In the multi-model approach, separate sub-models determine the consecutive predictions. Only one model is used in the factorised approach, but it is not recommended for long-range prediction. Since the cited multi-model and the factorised structure are nonlinear, successive online linearisation is necessary to obtain quadratic optimisation MPC problems. The relevant procedure for the multi-model comprised of neural networks of the Multi-Layer Perceptron (MLP) is discussed in [21] and the solution for the factorised MLP model is presented in [18].

It is true that convexity of the MPC optimisation task is guaranteed when a nonlinear model, including a neural network, is successively linearised online at the current operating point of the process [21,25,31]. Let us recall that MPC algorithms with successive linearisation dominate in practical applications [41]. Unfortunately, accuracy of the time-varying linearised model cannot be assessed beforehand. Possible inaccuracy of the linearised model badly influences the quality of predictions and the resulting control quality of MPC [21,25]. The problem does not exist when convex neural networks are used because the model is used for prediction without any linearisation, simplification and truncation, but convexity is still guaranteed.

State-space fundamental models may be used in MPC [28,41], but they require a description of technological processes using first principles. Conversely, the discussed convex neural models are developed using some data sets; no fundamental knowledge is necessary. The use of fundamental nonlinear models results in the same problems as in the case of other nonlinear models, i.e. the MPC optimisation task is nonlinear and when the model is linearised online, the mismatch between the original nonlinear model and its linear approximation is inevitable.

### 1.3. Contribution

This work presents a nonlinear MPC algorithm that uses fully and partially ICNNs models. Unlike the previous works concerned with convex neural networks [4,7,32,45], the model is deliberately not used recurrently. Instead, the multi-model approach is utilised, i.e. a set of convex models is used; they determine predictions for the consecutive sampling instants. The presented solution has two general advantages. Firstly, using input convex networks, the MPC optimisation problem is convex; there are no problems with multiple shallow local minima. Secondly, recurrence is unnecessary due to the multi-model structure, which means that the problem with accumulation of the prediction errors does not exist.

Four model structures for a polymerisation reactor benchmark process are considered: the linear multi-model, the full ICNN multi-model and the partially ICNN multi-model in two versions. In the case of the nonlinear neural models, determination of the best number of hidden nodes is discussed. Next, all models are used in MPC and the influence of model accuracy on control quality is discussed. Two simulation scenarios are considered for all model structures: a disturbance-free case and a more interesting situation when the process is affected by external time-varying disturbances. Finally, the influence of the model structure and the horizons used in MPC on the calculation time is discussed.

This work is organised in the following way. Section 2 defines the rudimentary MPC optimisation task. Next, Section 3 describes the classical fully and partially ICNN models, conditions necessary for model convexity are formulated and selection of the activation function is discussed. The main part of the article is given in Section 4 that describes the prediction method and the MPC algorithm based on fully and partially ICNN multi-models. Section 5 presents

and thoroughly discusses simulation results. Finally, Section 6 summarises the whole work; advantages and limitations of the presented approach are pointed out.

## 2. Problem formulation

### 2.1. Definition of the Basic MPC Optimisation Task

In this work,  $u$  and  $y$  stand for the manipulated and controlled variables, respectively. In MPC [6,28,41], a set of current and future values of the manipulated variable are calculated at each sampling instant,  $k = 0, 1, 2, \dots$

$$\mathbf{u}(k) = [u(k|k) \dots u(k + N_u - 1|k)]^T \quad (1)$$

The symbol  $k + p|k$  refers to the future sampling instant  $k + p$  considered at the current sampling  $k$ . In some versions of MPC, the decision vector is comprised of current and future changes of the manipulated variable, not their values. The length of the decision variable vector (1) is defined by the control horizon denoted by  $N_u$ . The current and future control policy is determined online as a result as an optimisation procedure. Typically, the minimised MPC cost function is

$$J(k) = \sum_{p=1}^N (y^{sp}(k + p|k) - \hat{y}(k + p|k))^2 + \lambda \sum_{p=0}^{N_u-1} (\Delta u(k + p|k))^2 \quad (2)$$

The set-point value of the controlled variable for the sampling instant  $k + p$  known at the current instant  $k$  is denoted by  $y^{sp}(k + p|k)$ . The predicted value of the controlled variable for the sampling instant  $k + p$  calculated at the instant  $k$  is  $\hat{y}(k + p|k)$ . Hence, the first part of the MPC cost function measures the squared control errors predicted over the prediction horizon  $N$ . The second part of the MPC cost function measures squared changes of the manipulated variable over the control horizon. It is a penalty term with a weight  $\lambda > 0$  used to minimise unwanted abrupt changes of that variable; the changes are defined as  $\Delta u(k + p|k) = u(k + p|k) - u(k + p - 1|k)$ . MPC optimisation is usually performed subject to some constraints that result from physical limitations of the hardware used. The classical MPC optimisation task is

$$\begin{aligned} & \min_{\mathbf{u}(k)} \{J(k)\} \\ & \text{subject to} \\ & u^{\min} \leq u(k + p|k) \leq u^{\max}, \quad p = 0, \dots, N_u - 1 \\ & \Delta u^{\min} \leq \Delta u(k + p|k) \leq \Delta u^{\max}, \quad p = 0, \dots, N_u - 1 \\ & y^{\min} \leq \hat{y}(k + p|k) \leq y^{\max}, \quad p = 1 \end{aligned} \quad (3)$$

In general, three types of constraints may be considered. Firstly, the magnitude of the manipulated variable may be limited;  $u^{\min}$  and  $u^{\max}$  stand for minimal and maximal magnitude, respectively. Secondly, the rate of change of the manipulated variable may be constrained;  $\Delta u^{\min}$  and  $\Delta u^{\max}$  stand for maximal negative and maximal positive changes, respectively. Thirdly, the predicted value of the process output may be limited;  $y^{\min}$  and  $y^{\max}$  stand for minimal and maximal magnitude, respectively. When the MPC optimisation task (3) is solved, the first element of the optimal vector (1) is applied to the process. The optimisation procedure is repeated at the next sampling instants.

## 3. Input convex neural networks

Let us recall equations that define the fully input convex neural network and the partially input convex architecture [1].

### 3.1. Fully input convex neural network

The fully input convex neural network is described by the general relation

$$y = f(x)$$

We assume that  $x \in \mathbb{R}^{n_x}$  and  $y \in \mathbb{R}$ .  $L$  stands for the number of layers. The output vector of the consecutive layers is defined by

$$z_{i+1} = \varphi_i(W_i^{(z)} z_i + W_i^{(x)} x + b_i) \quad (5)$$

where  $i = 0, \dots, L-1$ . The weights are denoted by  $W_i^{(z)}$  and  $W_i^{(x)}$ , the symbol  $b_i$  stands for bias coefficients.  $W_0^{(z)} = 0$  and  $\varphi_i$  stands for the nonlinear activation function of the  $i$ th layer, respectively. The last layer of the network has one linear node which means that the output is

$$y = z_L \quad (6)$$

It is recommended to calculate the network output from Eq. (5) using vector–matrix representation, not scalars. A very computationally efficient vector–matrix calculations are possible, e.g., in MATLAB.

As proved in [1], the considered neural network is convex in terms of the input vector  $x$  provided that all weights  $W_1^{(z)}, \dots, W_{L-1}^{(z)}$  are non-negative and all activation functions  $\varphi_i$  are convex and non-decreasing.

In order to evaluate convex properties of the described convex neural network, let us consider two static nonlinear functions. The first of them is the  $\mathbb{R} \rightarrow \mathbb{R}$  function

$$f_1(x) = x^2 + 10 \sin(2x) \quad (7)$$

the second one is the  $\mathbb{R}^2 \rightarrow \mathbb{R}$  function

$$f_2(x) = x_1^2 + x_2^2 + 10 \sin(x_1) + 10 \sin(x_2) \quad (8)$$

Both functions have multiple local minima. In general, they are not convex. A three-layer convex neural network is used to approximate the above functions. The first two layers have five hidden nodes of the ReLU type, while the last layer is linear. Figs. 1 and 2 compare the static functions (the validation data set) and outputs of convex neural networks. In both cases, the neural approximation is convex.

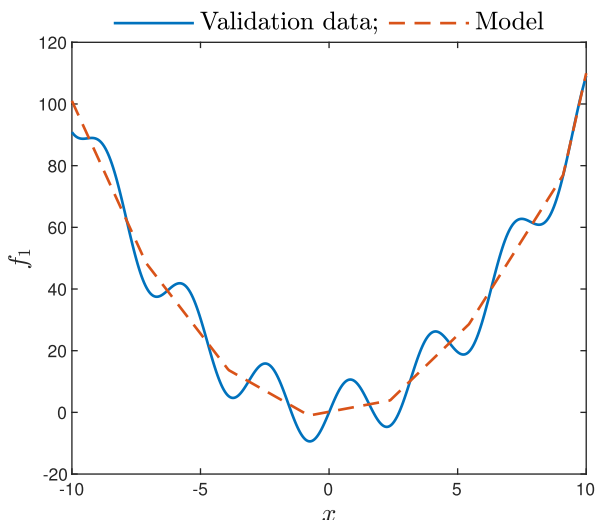


Fig. 1. Approximation of the static function (7): the validation data set vs. the output of the fully input convex neural network.

The obtained result needs a comment. On the one hand, it is expected that the resulting approximation maps provided by the neural networks are convex. It means that the approximator convexifies all local non-convexities. Such an approach works very well when the approximated function is generally convex with some minor local non-convex parts. On the other hand, the fully input convex neural network does not work when the approximated function is generally non-convex.

### 3.2. Partially input convex neural network

The fully input convex neural network models may be too restrictive which means that the resulting approximation accuracy may be below expectations. Hence, in the case of the partially input convex structure, we require that the model is convex in terms of a part of arguments. However, it may be not convex in terms of the rest of them. The general relation describes the partially input convex neural network

$$y = f(v, x) \quad (9)$$

We assume that  $v \in \mathbb{R}^{n_v}$ ,  $x \in \mathbb{R}^{n_x}$  and  $y \in \mathbb{R}$ . The network is required to be convex in terms of  $x$  but not convex in  $v$ .  $L$  stands for the number of layers. There are two types of hidden units: they are responsible for the  $v$ -path and the  $x$ -path, respectively. The output vector of the consecutive layers related to the  $v$ -path is

$$u_{i+1} = \tilde{\varphi}_i(\tilde{W}_i u_i + \tilde{b}_i) \quad (10)$$

where  $i = 0, \dots, L-1$ ,  $u_0 = v$  and  $\tilde{\varphi}_i$  stands for the nonlinear activation function of the  $i$ th layer, respectively. The weights are denoted by  $\tilde{W}_i$ , the symbol  $\tilde{b}_i$  stands for bias coefficients. The output vector of the consecutive layers related to the  $x$ -path is

$$z_{i+1} = \varphi_i(W_i^{(z)}(z_i \circ [W_i^{(zu)} u_i + b_i^{(z)}]) + W_i^{(x)}(x \circ (W_i^{(xu)} u_i + b_i^{(x)} + W_i^{(u)} u_i + b_i))) \quad (11)$$

where  $i = 0, \dots, L-1$ ,  $u_0 = v$ ,  $\varphi_i$  stands for the nonlinear activation function of the  $i$ th layer and  $\circ$  represents the Hadamard product, respectively. The weights are denoted by  $W_i^{(z)}$ ,  $W_i^{(zu)}$ ,  $W_i^{(x)}$ ,  $W_i^{(xu)}$  and  $W_i^{(u)}$ , the symbols  $b_i^{(z)}$ ,  $b_i^{(x)}$  and  $b_i$  stand for bias coefficients. The last layer of the network has one linear node which means that the output is given by Eq. (6). It is recommended to calculate the network output from Eq. (11) using vector–matrix representation that are possible, e.g., in MATLAB.

As proved in [1], the considered neural network is convex in terms of the input vector  $x$  provided that all weights  $W_1^{(z)}, \dots, W_{L-1}^{(z)}$  are non-negative and all activation functions  $\varphi_i$  are convex and non-decreasing.

### 3.3. Selection of activation functions

Selection of activation functions used in convex neural networks is crucial since for convexity of the model, it is required that all activation functions are convex and non-decreasing. Fig. 3 depicts 11 popular activation functions used in machine learning applications.

At first, let us consider the classical sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (12)$$

and the hyperbolic tangent one

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (13)$$

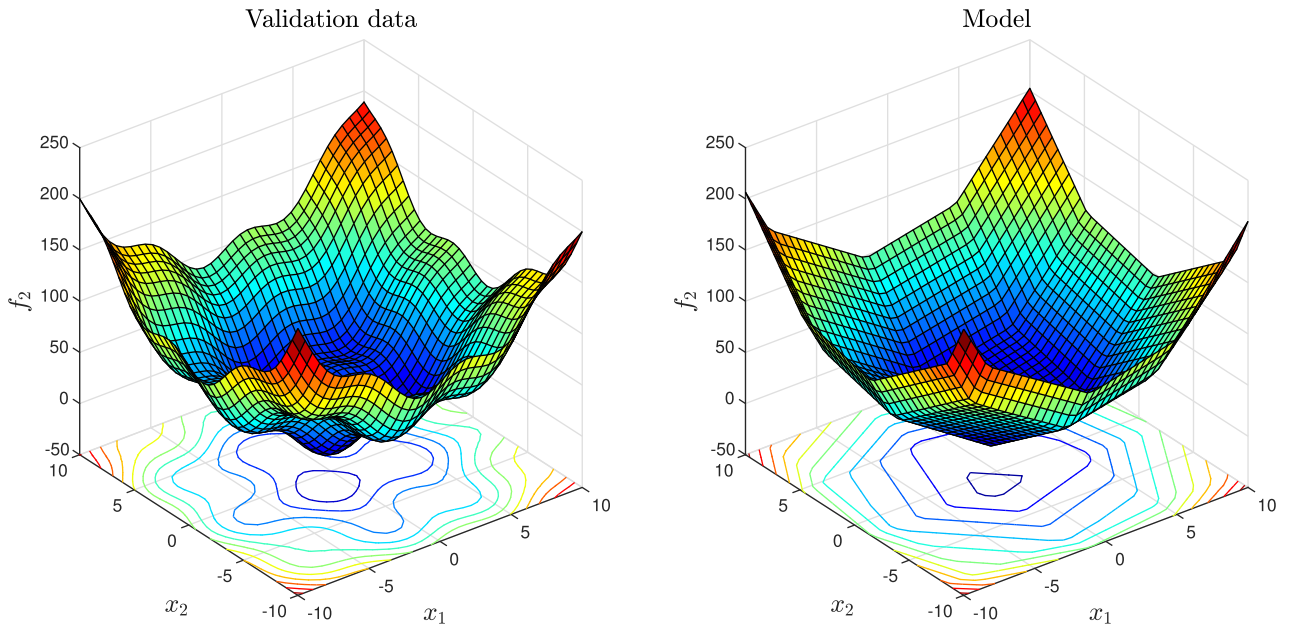


Fig. 2. Approximation of the static function (8): the validation data set vs. the output of the fully input convex. neural network.

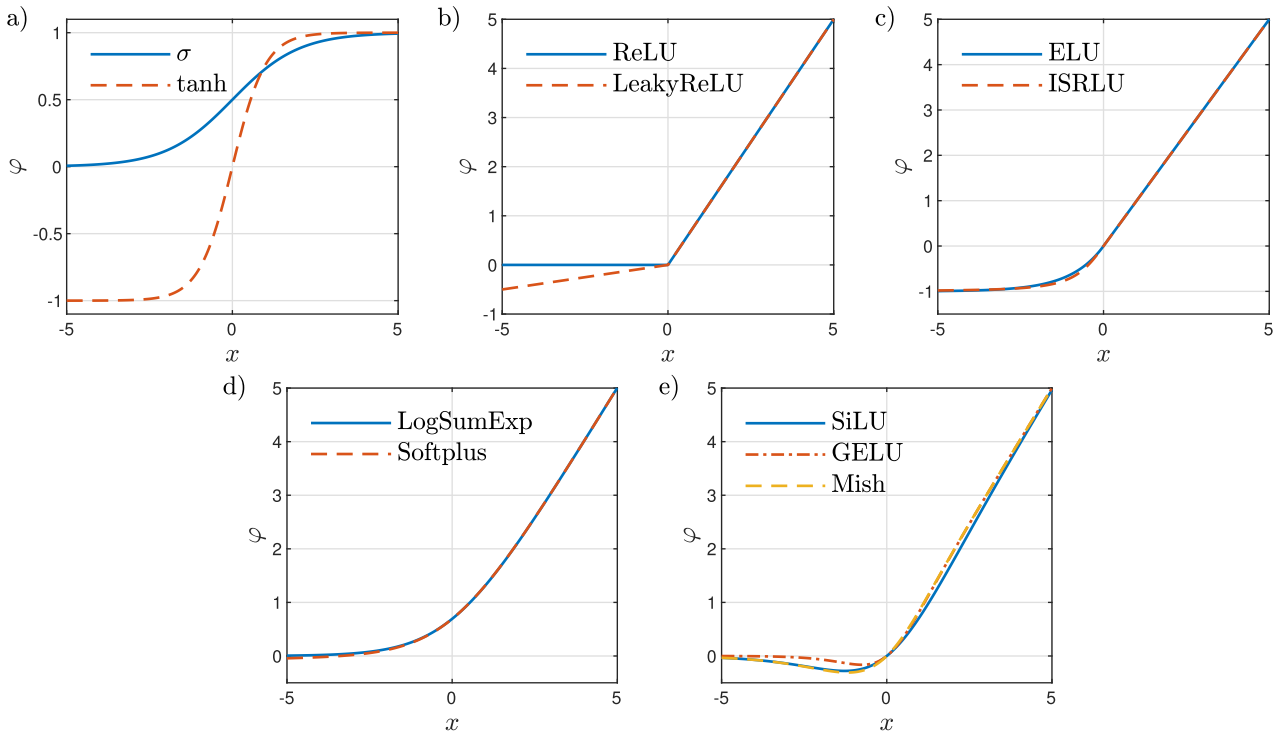


Fig. 3. The activation functions.

These functions are shown in Fig. 3a. Both functions are non-decreasing. They are convex for  $x \in \mathbb{R}^-$  but concave for  $x \in \mathbb{R}^+$ . As a result, they cannot be used in convex neural networks.

Next, we consider the Rectified Linear Unit (ReLU) function

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \quad (14)$$

and the Leaky Rectified Linear Unit (LeakyReLU) one

$$\text{LeakyReLU}(x) = \max(\alpha x, x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \quad (15)$$

These functions are shown in Fig. 3b; the parameter  $\alpha = 0.1$  is used in the case of the LeakyReLU function. These functions have two essential advantages: simplicity of calculation and elimination of the vanishing gradient problem that occurs in the case of the classical sigmoid and hyperbolic tangent functions (the gradient

frequently saturates which leads to very slow training). The ReLU and LeakyReLU functions are convex and non-decreasing.

Fig. 3c depicts the following functions: the Exponential Linear Unit (ELU) one

$$\text{ELU}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \quad (16)$$

and the Inverse Square Root Linear Unit (ISRLU) function

$$\text{ISRLU}(x) = \begin{cases} \frac{x}{\sqrt{1+2\alpha^2}} & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \quad (17)$$

where  $\alpha = 1$ . The parameter  $\alpha$  of ELU and ISRLU functions has been adjusted in such a way that they have similar shape. Both functions are supposed to be continuous versions of the ReLU function. Both functions are convex and non-decreasing.

Next, Fig. 3d shows another continuous versions of the ReLU function: Softplus

$$\text{Softplus}(x) = \ln(1 + e^x) \quad (18)$$

and LogSumExp

$$\text{LogSumExp}(x) = \ln(e^{2x} + e^x) \quad (19)$$

Both functions are convex and non-decreasing.

Finally, Fig. 3e depicts the following functions: Sigmoid Linear Unit (SiLU)

$$\text{SiLU}(x) = x\sigma(x) = \frac{x}{1 + e^{-x}} \quad (20)$$

Gaussian Error Linear Unit (GELU)

$$\text{GELU}(x) = \frac{x}{1 + e^{-1.702x}} \quad (21)$$

and Mish

$$\text{Mish}(x) = x \tanh(\ln(1 + e^x)) \quad (22)$$

We can see that all these functions are decreasing for  $x \in \mathbb{R}^-$ . Hence, they cannot be used in convex neural networks.

The following functions are non-decreasing and convex: ReLU, LeakyReLU, ELU, ISRLU, LogSumExp and Softplus. It means that each of them may be used and model convexity is guaranteed. However, one additional aspect has to be considered, namely the calculation time. In particular, it is enormously important in training since, during the training process, the model output is calculated numerous times in the consecutive training epochs. Let us give calculation times (scaled) obtained in MATLAB. Let the calculation time of the ReLU function be equal to 100%. The time increases moderately to approximately 150% for the LeakyReLU function, to some 500% for ELU and ISRLU functions, to some 1700% and 2000% for Softplus and LogSumExp functions, respectively. It is evident that ReLU and LeakyReLU functions have a major advantage compared to other functions: simplicity and fast calculation. Since the ReLU function may suffer from the “dead neuron” phenomenon (the neuron output signal is 0 for any  $x < 0$ ), the LeakyReLU function is used in this work and recommended.

#### 4. MPC algorithm based on input convex neural multi-models

Let us consider an example dynamical model described by the classical discrete-time equation

$$y(k) = f(u(k-1), \dots, u(k-n_B), y(k-1), \dots, y(k-n_A)) \quad (23)$$

where the function  $f: \mathbb{R}^{n_A+n_B+1} \rightarrow \mathbb{R}$  is nonlinear, e.g. represented by a neural network of a chosen structure. The MPC cost function (2) uses the predictions of the controlled variable that must be calcu-

lated from the model. For the example model (23), the consecutive predictions are

$$\hat{y}(k+1|k) = f(u(k|k), u(k-1), \dots, u(k-n_B+1), y(k), \dots, y(k-n_A+1)) + d(k) \quad (24)$$

$$\hat{y}(k+2|k) = f(u(k+1|k), u(k|k), u(k-1), \dots, u(k-n_B+2), \hat{y}(k+1|k), y(k), \dots, y(k-n_A+2)) + d(k) \quad (25)$$

$$\hat{y}(k+3|k) = f(u(k+2|k), u(k+1|k), u(k|k), u(k-1), \dots, u(k-n_B+3), \hat{y}(k+2|k), \hat{y}(k+1|k), y(k), \dots, y(k-n_A+3)) + d(k) \quad (26)$$

where  $d(k)$  stands for an estimation of the unmeasured disturbance that acts on the process output at the current sampling instant  $k$ . We can easily make four observations. Firstly, the predictions (24)–(26) depend on the current and future values of the manipulated variables, i.e.  $u(k|k), u(k+1|k), \dots$ ; they comprise the decision vector  $\mathbf{u}(k)$  (Eq. (1)) that is found at each sampling instant of MPC. The decision vector is calculated on-line as the result of solving the MPC optimisation problem (3). Secondly, the predictions (24)–(26) depend on some values of the manipulated variable calculated and applied to the process at the previous sampling instants are necessary, i.e.  $u(k-1), u(k-2), \dots$ . Thirdly, some measured values of the controlled variable for the current and previous sampling instants are necessary, i.e.  $y(k), y(k-1), \dots$ . Finally, the most important observation is the fact that the classical dynamical model (23) is used recurrently. More precisely, the prediction  $\hat{y}(k+1|k)$  is in fact a one-step-ahead prediction, while the prediction  $\hat{y}(k+2|k)$  depends recurrently on the prediction  $\hat{y}(k+1|k)$ , the prediction  $\hat{y}(k+3|k)$  depends recurrently on the predictions  $\hat{y}(k+1|k)$  and  $\hat{y}(k+2|k)$  etc.

Let us emphasise that recurrent model training (identification) is much more computationally demanding and time-consuming than obtaining the simple one-step-ahead predictor. Moreover, the resulting models are susceptible to noise and improper selection of the model order of dynamics; prediction errors are propagated. These factors may lead to inaccurate predictions and low control quality of MPC. Finally, the consecutive predictions depend in a complex, nonlinear way on the calculated decision vector (1). As a result, the MPC optimisation problem may be multi-modal and non-convex.

In order to eliminate the need of recurrent training and recurrent model utilisation in MPC, the multi-model approach may be used [21]. In this modelling methodology, independent sub-models are used for each sampling instant within the prediction horizon. It means that as many as  $N$  sub-models are necessary, i.e. the number of sub-models is defined by the prediction horizon used in MPC. Selection of appropriate horizons is discussed in classical books, e.g. [28,41]. In short, the prediction horizon should be long enough to obtain good control quality. On the other hand, long horizons lead to complicated models and longer calculation time. Hence, the horizon should be selected as a result of a compromise between control quality and model complexity. The essential advantage of the multi-model is the fact that all sub-models are trained independently as one-step-ahead non-recurrent models, computationally demanding recurrent training is not used. The consecutive sub-models are represented by the following functions

$$\hat{y}(k+1|k) = f_1(u(k|k), \mathbf{x}_p(k)) + d(k+1|k) \quad (27)$$

$$\hat{y}(k+2|k) = f_2(u(k+1|k), u(k|k), \mathbf{x}_p(k)) + d(k+2|k) \quad (28)$$

$$\hat{y}(k+3|k) = f_3(u(k+2|k), u(k+1|k), u(k|k), \mathbf{x}_p(k)) + d(k+3|k); \quad (29)$$

The arguments of the predictors are chosen in the following way. The multi-model predictions depend on future calculated future values of the manipulated variable, exactly in the same way the classical predictions defined by Eqs. 24,25 do. Moreover, the multi-model predictions depend on the current operating point of the process that is defined by the following past measurements of the manipulated variables as well as the current and past values of the controlled variable

$$\mathbf{x}_p(k) = [u(k-1) \dots u(k-n_b) \ y(k) \ y(k-1) \dots y(k-n_A)]^T \quad (30)$$

where the index  $p$  refers to past signals. Let us introduce the vectors of future manipulated variables

$$\mathbf{x}_f(k+1|k) = u(k|k) \quad (31)$$

$$\mathbf{x}_f(k+2|k) = [u(k+1|k), u(k|k)]^T \quad (32)$$

$$\mathbf{x}_f(k+3|k) = [u(k+2|k), u(k+1|k), u(k|k)]^T \quad (33)$$

⋮

In general, for  $p = 1, \dots, N$ , we have

$$\mathbf{x}_f(k+p|k) = [u(k+p-1|k), \dots, u(k|k)]^T \quad (34)$$

Hence, the universal multi-model prediction equation is

$$\hat{y}(k+p|k) = y(k+p|k) + d(k+p|k) \quad (35)$$

for  $p = 1, \dots, N$ , where

$$y(k+p|k) = f_p(\mathbf{x}_f(k+p|k), \mathbf{x}_p(k)) \quad (36)$$

denotes the output of the  $p$ -th sub-model. Since for each sampling instant a different sub-model is used, independent disturbance estimations should be used. They are calculated as differences between the measured process output signal for the current sampling instant,  $y(k)$ , and outputs of the consecutive sub-models. From Eq. (36), we have

$$d(k+p|k) = y(k) - f_p(\mathbf{x}_f^d(k+p|k), \mathbf{x}_p^d(k+p|k)) \quad (37)$$

When the sub-models are used for estimation of the unmeasured disturbances at the time  $k$ , the indices in Eqs. (30) and (34) must be moved back  $p$  steps which gives the following vectors of arguments

$$\mathbf{x}_f^d(k+p|k) = [u(k-1), \dots, u(k-p)]^T \quad (38)$$

$$\mathbf{x}_p^d(k+p|k) = [u(k-1-p) \dots u(k-n_b-p) \ y(k-p) \dots y(k-n_A-p)]^T \quad (39)$$

For simplicity, let us rewrite the Eq. (37) as

$$d(k+p|k) = y(k) - f_p(\mathbf{x}_d(k+p|k)) \quad (40)$$

where the argument vector used for the sub-model  $p$  consists of past signals

$$\mathbf{x}_d(k+p|k) = \left[ (\mathbf{x}_f^d(k+p|k))^T (\mathbf{x}_p^d(k+p|k))^T \right]^T \quad (41)$$

The term “unmeasured disturbance” in MPC literature, e.g. [28,41], describes model inaccuracy and actual disturbances that act on the process. The unmeasured disturbance estimation must be considered during prediction calculation because, in practice, all models, even very good ones, have some errors. Neglecting the unmeasured disturbance estimation in the prediction equation would result in an unwanted steady-state error, i.e. the actual process output would reach a steady-state different from the required set-point.

Prediction calculation using the multi-model is depicted in Fig. 4. Let us note that each sub-model is used twice at each sampling instant  $k$ . The first time it is used to find from Eq. (36) the out-

put of the sub-model for the future sampling instant  $k+p|k$ . Next, the sub-model is used to assess from Eq. (40) the unmeasured disturbance  $d(k+p|k)$ .

Different approximators may be used as the sub-models. For example, MLP neural networks with tanh activation functions are considered in [21]. Unfortunately, such MLP networks are non-convex approximators. As a result, the consecutive predictions  $\hat{y}(k+1|k), \dots, \hat{y}(k+N|k)$  are nonlinear and possibly non-convex functions of the calculated decision vector of MPC (Eq. (1)). To simplify calculations, in this work, we use fully input convex neural networks and partially input convex neural networks to represent the functions  $f_p$ . In the first case, both future and past vectors, i.e.  $\mathbf{x}_f(k+p|k)$  and  $\mathbf{x}_p(k)$ , respectively, are delivered as the inputs of the network. As a result, the predictions of the controlled variables are convex functions of all arguments of the model as defined by Eqs. (34) and (30). Of course, it means that the MPC optimisation task with the cost function (2) is convex in terms of the decision vector, which is good. However, it is also convex in terms of the past vector  $\mathbf{x}_p(k)$ , which is likely to lower approximation accuracy of the sub-models. Hence, in the second case, we use partially input convex neural networks to represent the functions  $f_p$ . In order to obtain convex predictions and the MPC optimisation task, only the future vector  $\mathbf{x}_f(k+p|k)$  is delivered as the input vector  $x$  in Eq. (9) and the past vector is delivered as the input vector  $v$ . Hence, the network output is convex in terms of the vector  $\mathbf{x}_f(k+p|k)$ , but not convex in terms of the vector  $\mathbf{x}_p(k)$ . As a result, the influence of the future is modelled in a convex way, while the influence of the past is approximated in a possibly non-convex manner.

When fully or partially convex neural multi-models are used for prediction calculation, the predictions  $\hat{y}(k+1|k), \dots, \hat{y}(k+N|k)$  are convex in terms of the calculated decision vector  $\mathbf{u}(k)$  defined by Eq. (1). Because the minimised MPC cost function (2) is quadratic, it is also convex in terms of the decision vector. As far as the constraints possible in the MPC optimisation task (3) are concerned, we have to distinguish two cases. The limits imposed on the magnitude and the rate of change of the manipulated variables are purely linear in the decision vector. Conversely, the constraints put on the predicted values of the controlled variable are nonlinear but convex which is guaranteed by convexity of the model. As a result, if the fully or partially convex neural multi-models are used for prediction, we obtain a convex MPC optimisation problem with linear and (or) convex constraints. Such problems have only one minimum which is a global solution, no other local minima are possible. Therefore, it can be solved very efficiently using available gradient-based optimisation solvers.

It is also possible to consider purely linear sub-models. They result in linear predictions and a quadratic MPC optimisation function, which is convex. In such a case, all constraints are linear. Unfortunately, when the considered process has nonlinear properties, linear sub-models may suffer from low approximation accuracy and result in unsatisfactory control quality.

## 5. Simulations

### 5.1. Process description

Effectiveness of convex multi-modelling and MPC based on such models is evaluated for a polymerisation reactor benchmark process [11]. The process manipulated variable is the inlet initiator flow rate  $F_I$  ( $\text{m}^3 \text{h}^{-1}$ ), the controlled variable is the value of Number Average Molecular Weight (NAMW) of the product ( $\text{kg kmol}^{-1}$ ). The continuous-time fundamental model of the process presented in [11] is used as the simulated process. The model consists of four nonlinear ordinary differential equations

$$\frac{dC_m(t)}{dt} = - \left[ Z_p \exp\left(\frac{-E_p}{RT}\right) + Z_{im} \exp\left(\frac{-E_{im}}{RT}\right) \right] C_m(t) P_0(t) \quad (42)$$

$$- \frac{F(t)C_m(t)}{V} + \frac{F(t)C_{min}}{V} \quad (43)$$

$$\frac{dC_l(t)}{dt} = -Z_l \exp\left(\frac{-E_l}{RT}\right) C_l(t) - \frac{F(t)C_l}{V} + \frac{F_l(t)C_{lin}}{V} \quad (44)$$

$$\frac{dD_0(t)}{dt} = \left[ 0.5Z_{Tc} \exp\left(\frac{-E_{Tc}}{RT}\right) + Z_{Td} \exp\left(\frac{-E_{Td}}{RT}\right) \right] P_0^2(t)$$

$$+ Z_{fm} \exp\left(\frac{-E_{fm}}{RT}\right) C_m(t) P_0(t) - \frac{F(t)D_0(t)}{V} \quad (45)$$

where

$$P_0(t) = \sqrt{\frac{2f^* C_l(t) Z_l \exp\left(\frac{-E_l}{RT}\right)}{Z_{Td} \exp\left(\frac{-E_{Td}}{RT}\right) + Z_{Tc} \exp\left(\frac{-E_{Tc}}{RT}\right)}} \quad (46)$$

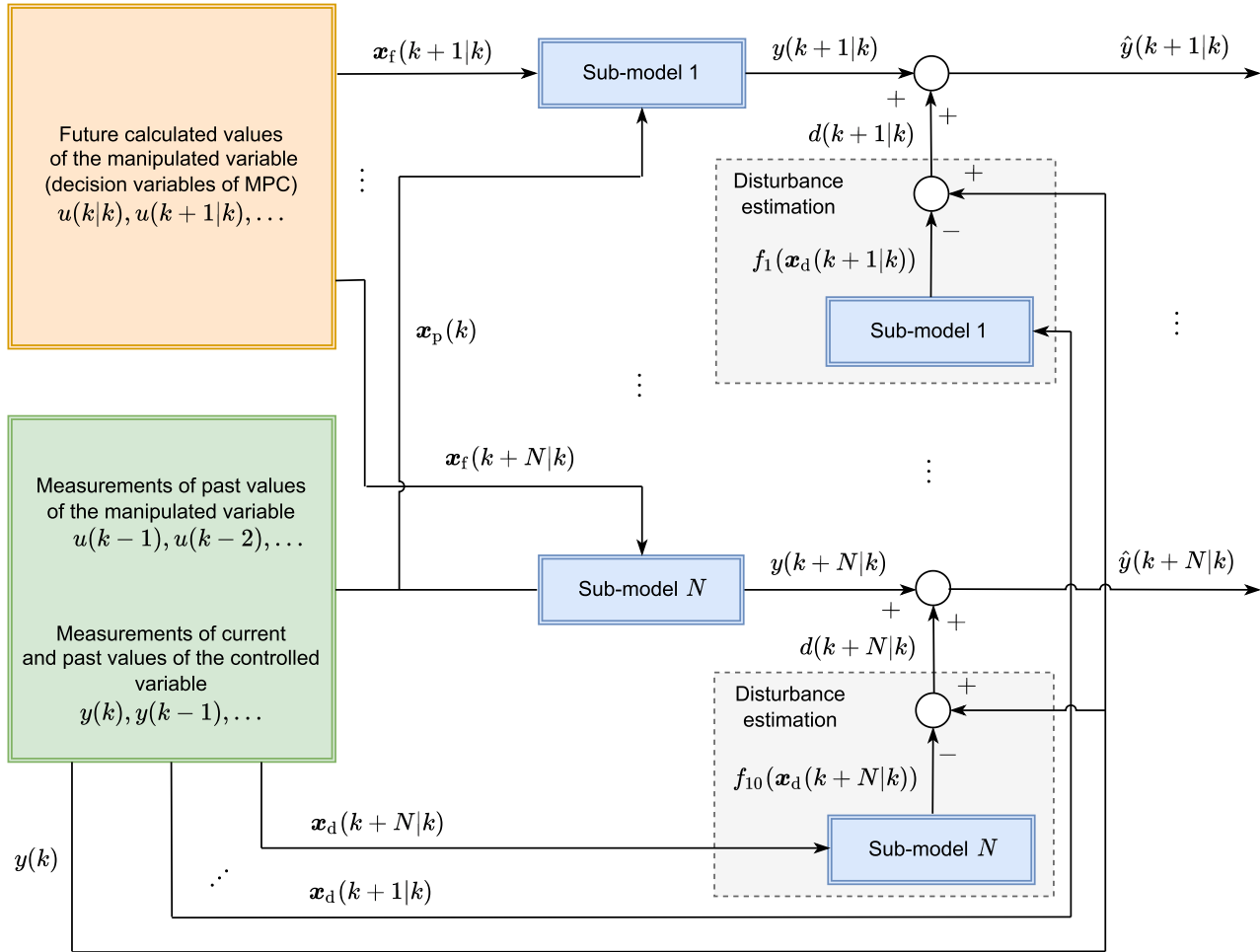


Fig. 4. Prediction calculation using the multi-model based on input convex neural networks.

Table 1

Parameters of the fundamental model of the polymerisation reactor.

| Parameter | Value                                     | Parameter | Value                                                                |
|-----------|-------------------------------------------|-----------|----------------------------------------------------------------------|
| $C_{lin}$ | $8 \text{ kmol m}^{-3}$                   | $R$       | $8.314 \text{ kJ kmol}^{-1} \text{ K}^{-1}$                          |
| $C_{min}$ | $6 \text{ kmol/m}^{-3}$                   | $T$       | $335 \text{ K}$                                                      |
| $E_{Tc}$  | $2.9442 \times 10^3 \text{ kJ kmol}^{-1}$ | $Z_{Tc}$  | $3.8223 \times 10^{10} \text{ m}^3 \text{ kmol}^{-1} \text{ h}^{-1}$ |
| $E_{Td}$  | $2.9442 \times 10^3 \text{ kJ kmol}^{-1}$ | $Z_{Td}$  | $3.1457 \times 10^{11} \text{ m}^3 \text{ kmol}^{-1} \text{ h}^{-1}$ |
| $E_{fm}$  | $7.4478 \times 10^4 \text{ kJ kmol}^{-1}$ | $Z_{fm}$  | $1.0067 \times 10^{15} \text{ m}^3 \text{ kmol}^{-1} \text{ h}^{-1}$ |
| $E_l$     | $1.2550 \times 10^5 \text{ kJ kmol}^{-1}$ | $Z_l$     | $3.7920 \times 10^{18} \text{ h}^{-1}$                               |
| $E_p$     | $1.8283 \times 10^4 \text{ kJ kmol}^{-1}$ | $Z_p$     | $1.7700 \times 10^9 \text{ m}^3 \text{ kmol}^{-1} \text{ h}^{-1}$    |
| $f^*$     | $0.58$                                    | $V$       | $0.1 \text{ m}^3$                                                    |
| $M_m$     | $100.12 \text{ kg kmol}^{-1}$             |           |                                                                      |

and one algebraic output equation

$$\text{NAMW} = \frac{D_1(t)}{D_0(t)} \quad (47)$$

Initial operating conditions are:  $F_1 = 0.028328 \text{ m}^3 \text{ h}^{-1}$ ,  $F = 1 \text{ m}^3 \text{ h}^{-1}$ ,  $\text{NAMW} = 20000 \text{ kg kmol}^{-1}$ ,  $C_m = 5.3745 \text{ kmol m}^{-3}$ ,  $C_1 = 2.2433 \times 10^{-1} \text{ kmol m}^{-3}$ ,  $D_0 = 3.1308 \times 10^{-3} \text{ kmol m}^{-3}$ ,  $D_1 = 6.2616 \times 10^{-1} \text{ kmol m}^{-3}$ . All parameters of the model are given in Table 1.

As a result of its nonlinear behaviour, the discussed polymerisation reactor is a typical benchmark used in modelling and control tasks. As far as a black-box approximation of the reactor is concerned, different model structures are possible: Volterra models [11], dynamical models obtained through genetic programming [29]; a comparison of black-box neural MLP models, neural Wiener and Hammerstein models are presented in [21].

As far as control of the reactor is concerned, several approaches are possible: an MPC algorithm based on a linear model with an auxiliary loop to capture process nonlinearity [11], an adaptive Internal Model Control (IMC) algorithms [9], an adaptive MPC strategy based on the just-in-time learning (JITL) technique [15], an adaptive MPC method based on orthogonal wavelet networks [44], offset-free state-space MPC algorithms based on the discrete-time fundamental model with state estimation [23,42], computationally efficient MPC algorithms with on-line linearisation in which MLP neural black-box, MLP neural Wiener and MLP neural Hammerstein models are used [21], Dynamic Matrix Control (DMC) algorithm based on step response MLP neural models [19].

The polymerisation reactor may be approximated for MPC using the MLP multi-model [21] or the factorised MLP model [18].

## 5.2. Process modelling for MPC

In this work, three classes of models are considered:

- a) the linear multi-model,
- b) the fully input convex neural multi-model,

c) the partially input convex neural multi-model.

Training of all models is performed off-line and the obtained models are also validated off-line. The whole training and validation data sets are depicted in Fig. 5; no other data sets are used. It is assumed that sufficiently reach training and validation data sets are available. Data sets may be recorded during process operation or from open-loop simulations (without any controller) of the fundamental model of the process. The data sets used in this work have been obtained from simulations of the fundamental model, which is defined by Eqs. (42)–(47). The sampling time is 1.8 s. Since the manipulated and controlled variables have different magnitude, they are scaled

$$u = 100(F_1 - \bar{F}_1), \quad y = 0.0001(\text{NAMW} - \bar{\text{NAMW}}) \quad (48)$$

where  $\bar{F}_1 = 0.016783$  and  $\bar{\text{NAMW}} = 20000$  denote values of process variables at the nominal operating point.

We assume the second order of model dynamics, i.e.  $n_A = n_B = 2$ . Taking into account Eq. (30), the past vector

$$\mathbf{x}_p(k) = [u(k-1) \ u(k-2) \ y(k) \ y(k-1) \ y(k-2)]^T \quad (49)$$

**Table 2**  
Errors of linear sub-models for training and validation data sets.

| $p$ | Number of parameters | $E_{\text{train}}$      | $E_{\text{val}}$        |
|-----|----------------------|-------------------------|-------------------------|
| 1   | 6                    | $4.7391 \times 10^{-1}$ | $5.5597 \times 10^{-1}$ |
| 2   | 7                    | $3.2125 \times 10^0$    | $3.6753 \times 10^0$    |
| 3   | 8                    | $8.9096 \times 10^0$    | $9.9670 \times 10^0$    |
| 4   | 9                    | $1.7215 \times 10^1$    | $1.8930 \times 10^1$    |
| 5   | 10                   | $2.7431 \times 10^1$    | $2.9798 \times 10^1$    |
| 6   | 11                   | $3.8827 \times 10^1$    | $4.1841 \times 10^1$    |
| 7   | 12                   | $5.0767 \times 10^1$    | $5.4437 \times 10^1$    |
| 8   | 13                   | $6.2734 \times 10^1$    | $6.7089 \times 10^1$    |
| 9   | 14                   | $7.4338 \times 10^1$    | $7.9413 \times 10^1$    |
| 10  | 15                   | $8.5308 \times 10^1$    | $9.1131 \times 10^1$    |

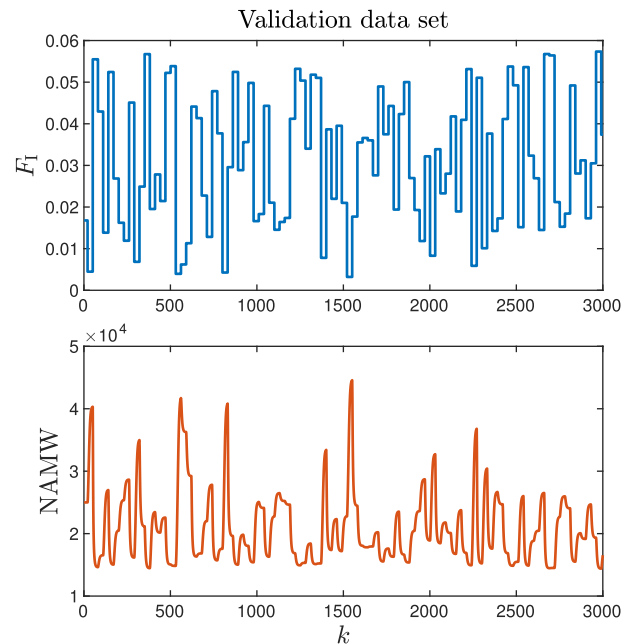
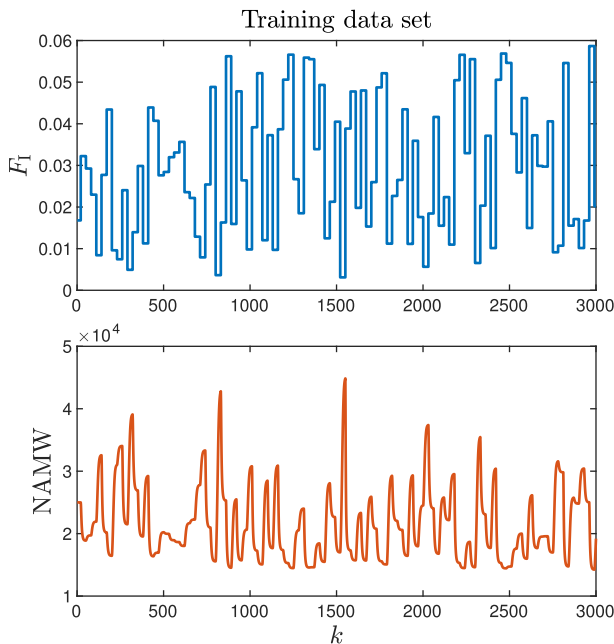


**Fig. 5.** Training and validation data sets.

is the same for all sampling instants within the prediction horizon. Conversely, from Eq. (34), it is clear that the future vector  $\mathbf{x}_i(k+p|k)$  depends on the number of the sub-model. We assume that the prediction horizon  $N = 10$ . It means that we have to develop 10 sub-models.

### 5.2.1. Linear multi-model

First, let us verify if the linear multi-model may be used for the polymerisation benchmark process. Model errors for training and validation data sets are given in Table 2; they are denoted by  $E_{\text{train}}$  and  $E_{\text{val}}$ , respectively. Two observations are possible. Firstly, the greater  $p$ , the lower model accuracy, i.e. model accuracy deter-

iorates for the sampling instants far from the current sampling instant. Secondly, the order of model errors is the same for two sets of data which means that model identification is performed correctly. In order to easily assess accuracy of all sub-models, let us consider Fig. 6 that depicts the relation between the validation data set vs. outputs of the consecutive linear sub-models. We may accept the accuracy of the first three or four sub-models, but the rest of them are imprecise. Fig. 7 compares the first 1000 samples of the validation data set vs. outputs of linear sub-models for  $p = 1$  and  $p = 10$ . The first sub-model is very good, while the last one is very imprecise; it cannot provide long-range prediction.

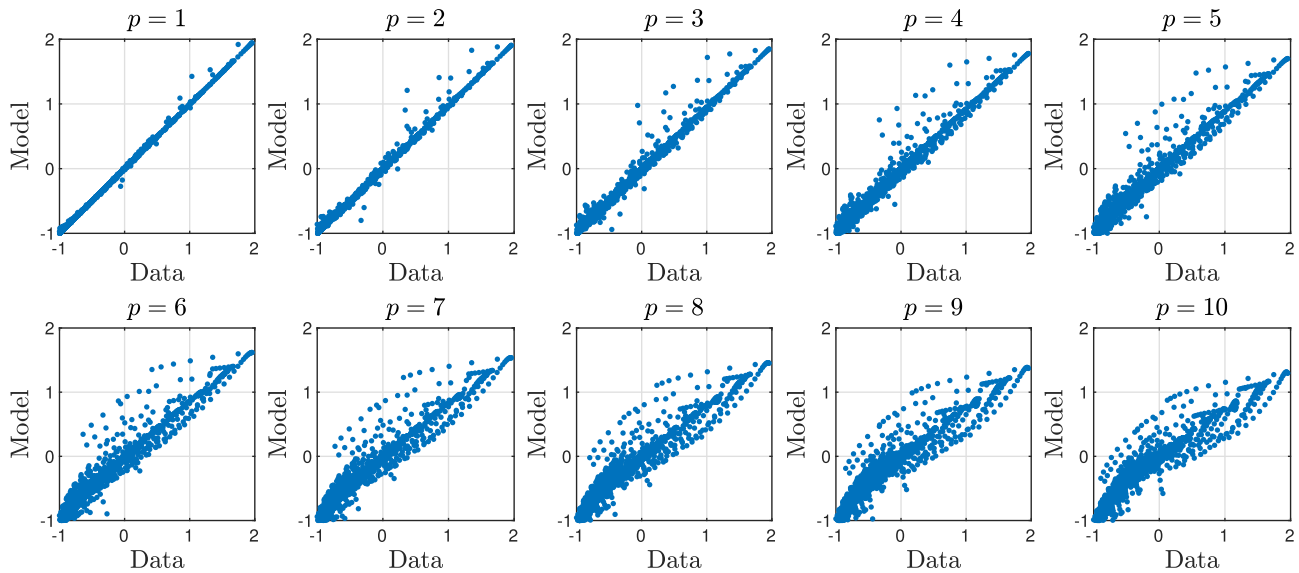


Fig. 6. The relation between the validation data set vs. outputs of linear sub-models for  $p = 1, \dots, 10$ .

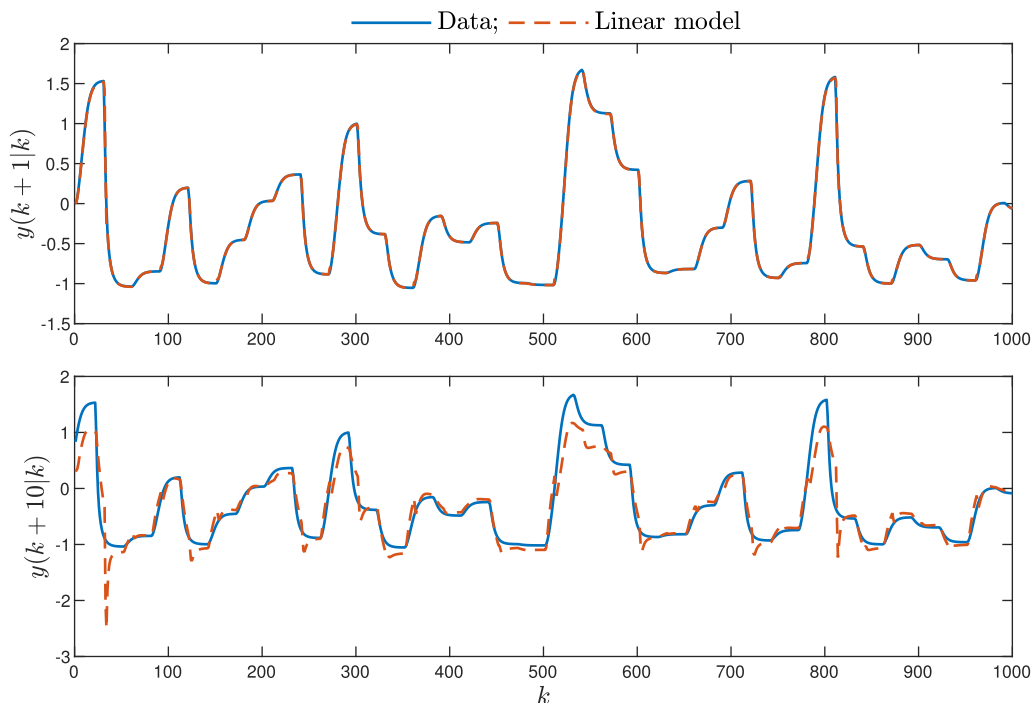


Fig. 7. The validation data set (the first 1000 samples) vs. outputs of linear sub-models for  $p = 1$  and  $p = 10$ .

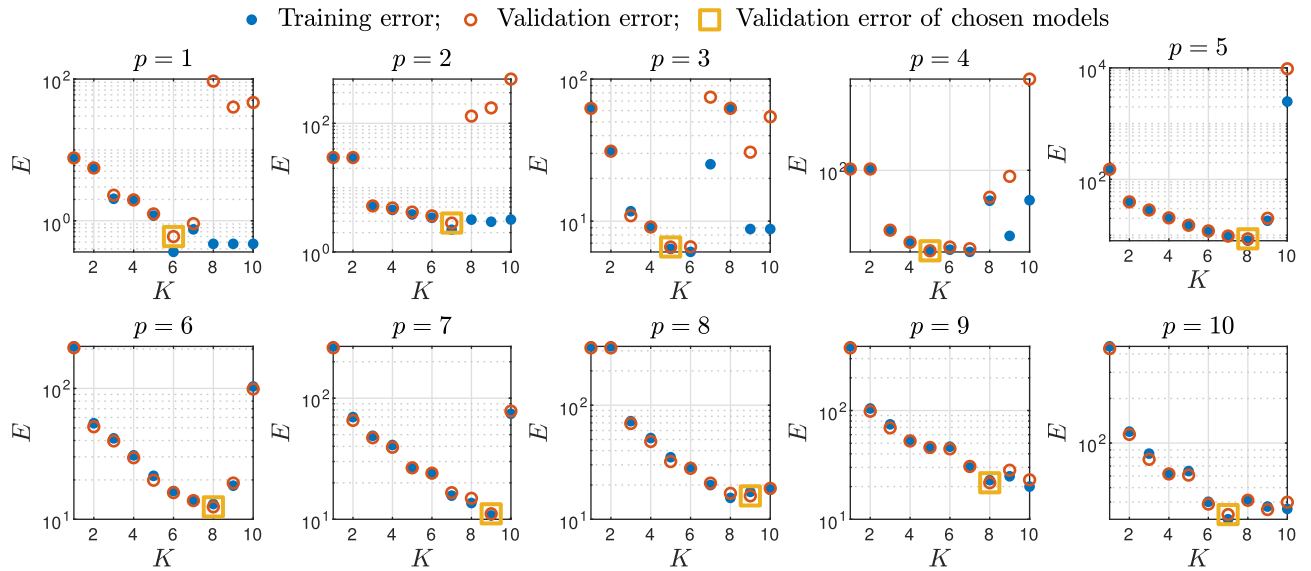


**Fig. 8.** The influence of the number of hidden nodes  $K$  on errors for training and validation data sets of fully input convex neural sub-models for  $p = 1, \dots, 10$ ; validation errors of the best sub-models are marked.

**Table 3**

Errors of the best fully input convex neural sub-models for training and validation data sets.

| $p$ | Number of nodes ( $K$ ) | Number of parameters | $E_{\text{train}}$      | $E_{\text{val}}$        |
|-----|-------------------------|----------------------|-------------------------|-------------------------|
| 1   | [6 6]                   | 133                  | $3.6292 \times 10^{-1}$ | $5.9880 \times 10^{-1}$ |
| 2   | [7 7]                   | 176                  | $2.2067 \times 10^0$    | $2.8145 \times 10^0$    |
| 3   | [5 5]                   | 129                  | $6.3731 \times 10^0$    | $6.5829 \times 10^0$    |
| 4   | [5 5]                   | 140                  | $1.1249 \times 10^1$    | $1.1166 \times 10^1$    |
| 5   | [8 8]                   | 259                  | $7.9324 \times 10^0$    | $8.5904 \times 10^0$    |
| 6   | [8 8]                   | 276                  | $1.3074 \times 10^1$    | $1.2443 \times 10^1$    |
| 7   | [9 9]                   | 337                  | $1.0743 \times 10^1$    | $1.1102 \times 10^1$    |
| 8   | [9 9]                   | 356                  | $1.7380 \times 10^1$    | $1.6039 \times 10^1$    |
| 9   | [8 8]                   | 327                  | $2.2865 \times 10^1$    | $2.1674 \times 10^1$    |
| 10  | [7 7]                   | 296                  | $3.0585 \times 10^1$    | $3.2916 \times 10^1$    |

### 5.2.2. Fully input convex neural multi-models

Next, we study effectiveness of fully input convex neural multi-models. At first, we have to find the necessary number of hidden layers and the number of nodes in each layer. The LeakyReLU activation function is used. Unfortunately, large errors are obtained when only one hidden layer is used, even for a high number of hidden nodes. That is why neural networks with two hidden layers are considered. For simplicity, we assume that the same number of hidden nodes, denoted by  $K$ , is used in both hidden layers. Fig. 8 shows the influence of the number of hidden nodes  $K$  on errors for training and validation data sets of fully input convex neural sub-models for  $p = 1, \dots, 10$ . In order to select the sub-models which are next used in MPC, we consider the validation error only, i.e. we choose the sub-models for which the validation error has the lowest value. The chosen sub-models have 5, 6, 7, 8 or 9 nodes in two hidden layers. When the number of hidden nodes is increased, the training error decreases while the validation error

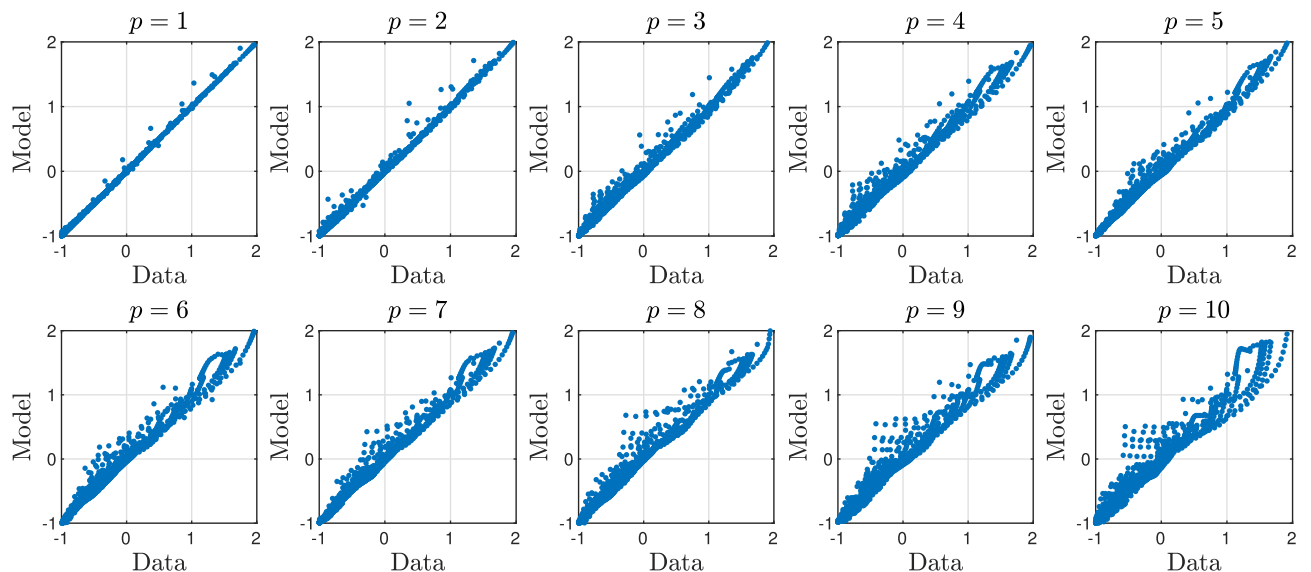


**Fig. 9.** The relation between the validation data set vs. outputs of the best fully input convex neural sub-models for  $p = 1, \dots, 10$ .

increases. It means that we cannot use too many hidden nodes. The application of a much more complicated network with three hidden layers does not improve accuracy.

For the chosen sub-model architectures, model errors for training and validation data sets are given in Table 3; the total number of parameters is also specified. Fig. 9 depicts the relation between the validation data set vs. outputs of the consecutive neural sub-models. When compared with the linear multi-model (Table 2 and Fig. 6), we can see that the neural networks result in lower errors. Namely, for  $p = 1, 2, 3$ , the neural sub-models have slightly lower errors, but for  $p \geq 4$  the errors are 2–3 times lower. Similarly to the linear sub-models, the greater  $p$ , the lower model accuracy

in the case of the neural ones. Fig. 10 compares the first 1000 samples of the validation data set vs. outputs of fully input convex neural sub-models for  $p = 1$  and  $p = 10$ . When compared with Fig. 7 that presents a similar comparison for linear sub-models, we see that approximation accuracy at the end of the prediction horizon is better in comparison with the linear case but not excellent.

In general, it is essential to emphasise that the accuracy of the neural sub-models based on fully input convex neural networks is better than that of linear ones. However, neural sub-models are not excellent, particularly for long-range prediction. Furthermore, increasing the number of hidden layers does not improve accuracy of neural sub-models. The considered class of models

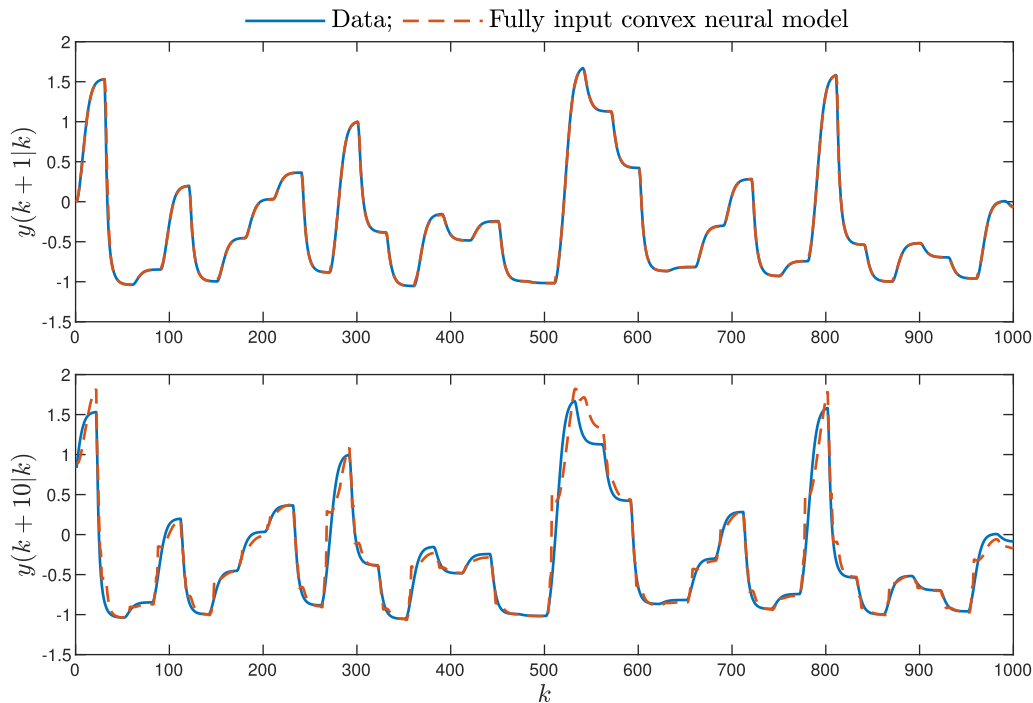


Fig. 10. The validation data set (the first 1000 samples) vs. outputs of fully input convex neural sub-models for  $p = 1$  and  $p = 10$ .

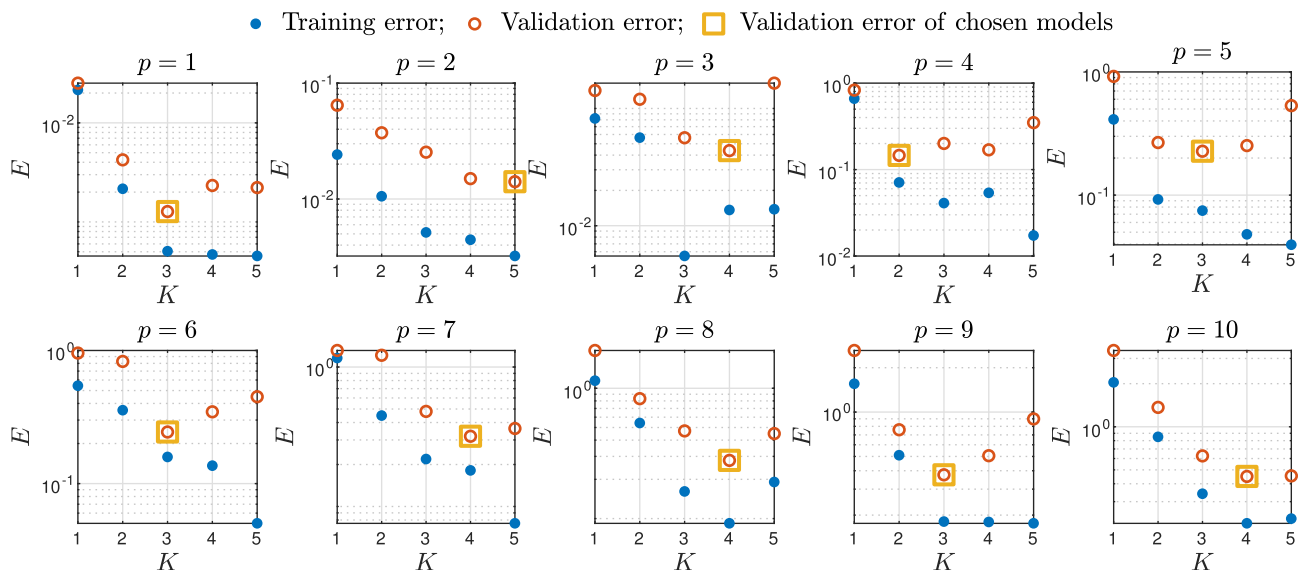


Fig. 11. The influence of the number of hidden nodes  $K$  on errors for training and validation data sets of partially input convex neural sub-models for  $p = 1, \dots, 10$ ; validation errors of the best sub-models are marked.

**Table 4**

Errors of the best partially input convex neural sub-models for training and validation data sets.

| $p$ | Number of nodes ( $K$ ) | Number of parameters | $E_{\text{train}}$      | $E_{\text{val}}$        |
|-----|-------------------------|----------------------|-------------------------|-------------------------|
| 1   | 3                       | 73                   | $5.0542 \times 10^{-4}$ | $1.2724 \times 10^{-3}$ |
| 2   | 5                       | 143                  | $3.2198 \times 10^{-3}$ | $1.4102 \times 10^{-2}$ |
| 3   | 4                       | 130                  | $1.3593 \times 10^{-2}$ | $4.3726 \times 10^{-2}$ |
| 4   | 2                       | 86                   | $7.0774 \times 10^{-2}$ | $1.4592 \times 10^{-1}$ |
| 5   | 3                       | 129                  | $7.4714 \times 10^{-2}$ | $2.2721 \times 10^{-1}$ |
| 6   | 3                       | 143                  | $1.5831 \times 10^{-1}$ | $2.4418 \times 10^{-1}$ |
| 7   | 4                       | 194                  | $1.8125 \times 10^{-1}$ | $3.1890 \times 10^{-1}$ |
| 8   | 4                       | 210                  | $9.1876 \times 10^{-2}$ | $2.7971 \times 10^{-1}$ |
| 9   | 3                       | 185                  | $1.8057 \times 10^{-1}$ | $3.7536 \times 10^{-1}$ |
| 10  | 4                       | 242                  | $2.1155 \times 10^{-1}$ | $4.5007 \times 10^{-1}$ |

has a structural limitation that is full convexity with respect to all model arguments.

### 5.2.3. Partially input convex neural multi-models

Finally, we concentrate on partially input convex neural multi-models. One hidden layer with the LeakyReLU activation function is used. Fig. 11 shows the influence of the number of hidden nodes  $K$  on errors for training and validation data sets of fully input convex neural sub-models for  $p = 1, \dots, 10$ . To choose the best sub-models which are next used in MPC, we consider the sub-models which yield the minimal validation error. The chosen sub-models have 3, 4 or 5 nodes in the hidden layer.

Errors of the best partially input convex neural sub-models for training and validation data sets are shown in Table 4; the total number of parameters is also specified. Fig. 12 depicts the relation between the validation data set vs. outputs of the consecutive partially input convex neural sub-models. When compared with the fully input convex neural sub-models (Table 3 and Fig. 9), we can see that the best partially input convex structures are many times more precise than corresponding fully input convex ones. The least favourable ratio is 34.8 (the sub-models for  $p = 7$ ) while the most favourable gain is 470.6 (the sub-models for  $p = 1$ ). Of course, we can see that the prediction error increases at the end of the prediction horizon. However, in the case of partially input convex sub-models, all errors are much lower than in the case of fully input

**Table 5**

Errors of the simplest partially input convex neural sub-models for training and validation data sets.

| $p$ | Number of nodes ( $K$ ) | Number of parameters | $E_{\text{train}}$      | $E_{\text{val}}$        |
|-----|-------------------------|----------------------|-------------------------|-------------------------|
| 1   | 1                       | 29                   | $2.1452 \times 10^{-2}$ | $2.5171 \times 10^{-2}$ |
| 2   | 1                       | 39                   | $2.4121 \times 10^{-2}$ | $6.4469 \times 10^{-2}$ |
| 3   | 1                       | 49                   | $8.2006 \times 10^{-2}$ | $1.4233 \times 10^{-1}$ |
| 4   | 1                       | 59                   | $6.5950 \times 10^{-1}$ | $8.3014 \times 10^{-1}$ |
| 5   | 1                       | 69                   | $4.1105 \times 10^{-1}$ | $9.2013 \times 10^{-1}$ |
| 6   | 1                       | 79                   | $5.4284 \times 10^{-1}$ | $9.5637 \times 10^{-1}$ |
| 7   | 1                       | 89                   | $1.1587 \times 10^0$    | $1.3140 \times 10^0$    |
| 8   | 1                       | 99                   | $1.1418 \times 10^0$    | $1.9499 \times 10^0$    |
| 9   | 1                       | 109                  | $1.5541 \times 10^0$    | $2.6174 \times 10^0$    |
| 10  | 1                       | 119                  | $2.0398 \times 10^0$    | $3.4100 \times 10^0$    |

convex structures. As model accuracy is very good, increasing the number of hidden layers is unnecessary.

It is interesting to analyse accuracy of the simplest partially input convex sub-models in which only one hidden node is used, i.e.  $K = 1$ . Table 5 presents training and validation errors of the consecutive sub-models; the total number of parameters is also specified. Fig. 13 depicts the relation between the validation data set vs. outputs of the simplest partially input convex neural sub-models for  $p = 1, \dots, 10$ . When compared with the results for the best partially input convex neural sub-models (Table 4, Fig. 12), the results are worse, but still good. In particular, prediction accuracy at the end of the prediction horizon is good. Furthermore, the partially input convex multi-models are much better than fully convex architectures (Fig. 9 and Table 3).

Let us consider Fig. 14 that compares the first 1000 samples of the validation data set vs. outputs of two considered versions of the partially input convex neural sub-models for  $p = 1$  and  $p = 10$ , i.e. the best ones and the simplest structures with only one hidden node. Accuracy of the best sub-models is excellent, but the simplest ones are only slightly worse. When compared with Fig. 10 that presents a similar comparison for fully input convex sub-models, we see that application of partially input convex networks is very beneficial since fully convex networks are too restrictive, which badly affects model accuracy.

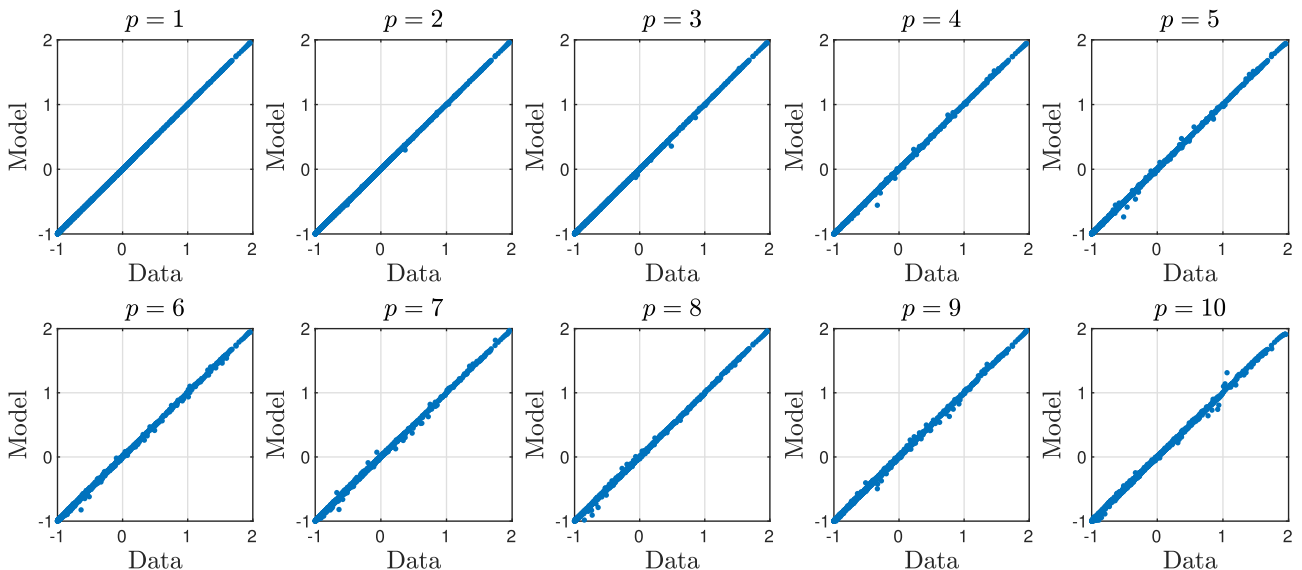


**Fig. 12.** The relation between the validation data set vs. outputs of the best partially input convex neural sub-models for  $p = 1, \dots, 10$ .

### 5.2.4. Model comparison

Having found four models, let us compare thoroughly their accuracy. Fig. 15 compares the models in terms of errors for training and validation data sets for the consecutive sub-models, i.e. for  $p = 1, \dots, 10$ . Of course, the models have to be compared for the validation error, but it is useful to find out that the order of models' errors is the same for both training and validation sets. It means that the number of training samples is sufficient and they are chosen properly.

We can observe that the greatest errors characterise the linear model. Interestingly, fully convex neural sub-models are slightly

better (i.e. model errors are lower) for  $p = 1, 2, 3, 4$  while this difference increases for  $p = 5, \dots, N$ . On the other hand, fully convex neural sub-models are the worst ones among the three considered nonlinear models. The simplest partially input convex neural sub-models are much better for all samplings instants over the prediction horizon. It means that fully convex models cannot precisely describe the properties of the considered process. The non-convex part of the sub-models turns out to be beneficial. Finally, the best partially input convex neural sub-models are better than the simplified ones. Interestingly, the gain of replacing fully convex models with the simplest partially convex ones is larger than the

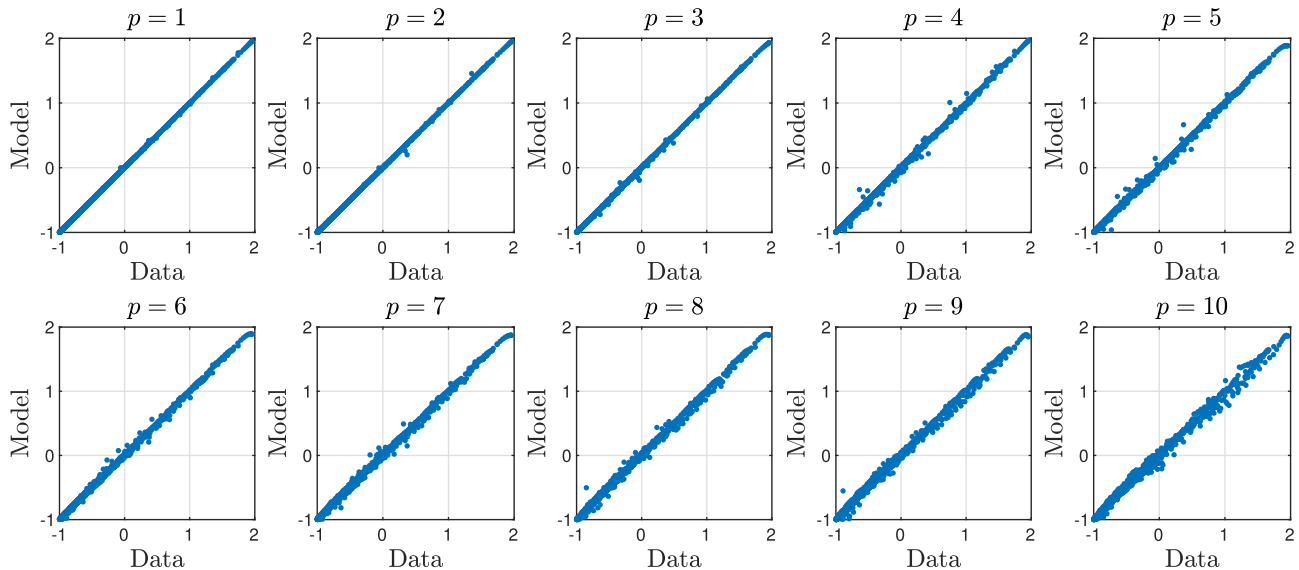


Fig. 13. The relation between the validation data set vs. outputs of the simplest partially input convex neural sub-models for  $p = 1, \dots, 10$ .

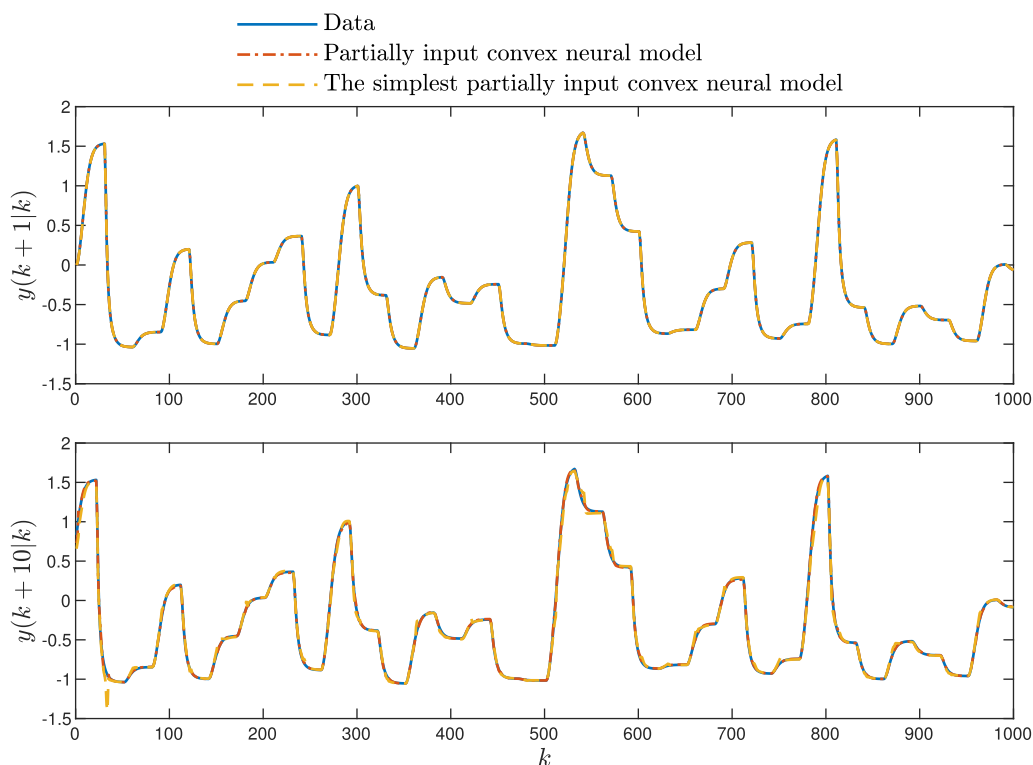


Fig. 14. The validation data set (the first 1000 samples) vs. outputs of the best and the simplest partially input convex neural sub-models for  $p = 1$  and  $p = 10$ .

difference between simplified and the best partially input convex models.

Fig. 16 depicts the number of parameters of the consecutive sub-models. Because fully input convex neural sub-models have two hidden layers, fully input structures are characterised by the largest number of parameters. Partially input convex structures have only one hidden layer, which results in a significantly lower number of model parameters. The simplest partially input convex neural networks require an even lower number of parameters.

The sub-models based on the simplest partially input convex neural networks are the most interesting since they have good accuracy and a moderate number of parameters.

### 5.3. Process control

#### 5.3.1. Comparison of control accuracy

MPC algorithms based on four types of models are considered:

- the linear multi-model,
- the fully input convex neural multi-model,
- the partially input convex neural multi-model,
- the simplest partially input convex neural multi-model.

All MPC algorithms have the same parameters:  $N = 10, N_u = 3, \lambda = 0.5$ . The constraints imposed on the manipulated variable are defined by:  $F_{l,\min} = 0.003, F_{l,\max} = 0.06$ . The tuning parameters are the same as those used in literature on MPC of the considered process [24].

At first, let us check if the MPC algorithms based on the linear model can control the process. Fig. 17 depicts simulation results for a set-point trajectory comprised of five steps. Unfortunately, the control quality is relatively low for four set-point changes since both manipulated and controlled variables exhibit oscillatory behaviour; the oscillations are damped but slowly. Oscillations are not observed for one set-point change, but control is very slow. Thus, the required set-point is never reached. The reason for unsatisfactory control quality is the low accuracy of the linear model, as depicted in Fig. 6.

Next, we compare effectiveness of MPC algorithms based on three nonlinear neural models. Simulation results are given in Fig. 18. We observe the following. Firstly, because accuracy of the fully input convex neural model is the lowest among all nonlin-

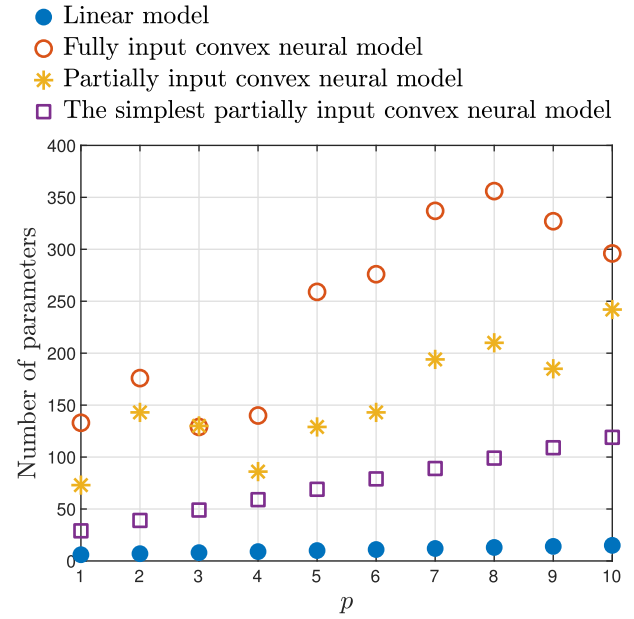


Fig. 16. Comparison of the number of parameters for four considered types of sub-models;  $p = 1, \dots, 10$ .

ear models, as shown in Fig. 9, the MPC algorithm that uses such a model for prediction gives the worst performance. Of course, the resulting trajectories are much better compared to the MPC scheme based on the linear model (Fig. 17). In the case of the fully input convex neural model, we have two disadvantages: significant overshoot present for the first and the last set-point changes and slow response for the fourth change.

Secondly, control quality is much better when the best partially input convex model is used since accuracy of the underlying model is much better, as depicted in Fig. 12. The overshoot is low, the regulation time is short and there are no oscillations.

Thirdly, when the simplest partially input convex model is used for prediction in MPC, the resulting control quality is very good, practically the same as in the case of much more complicated the best structures. However, although accuracy of the simplest

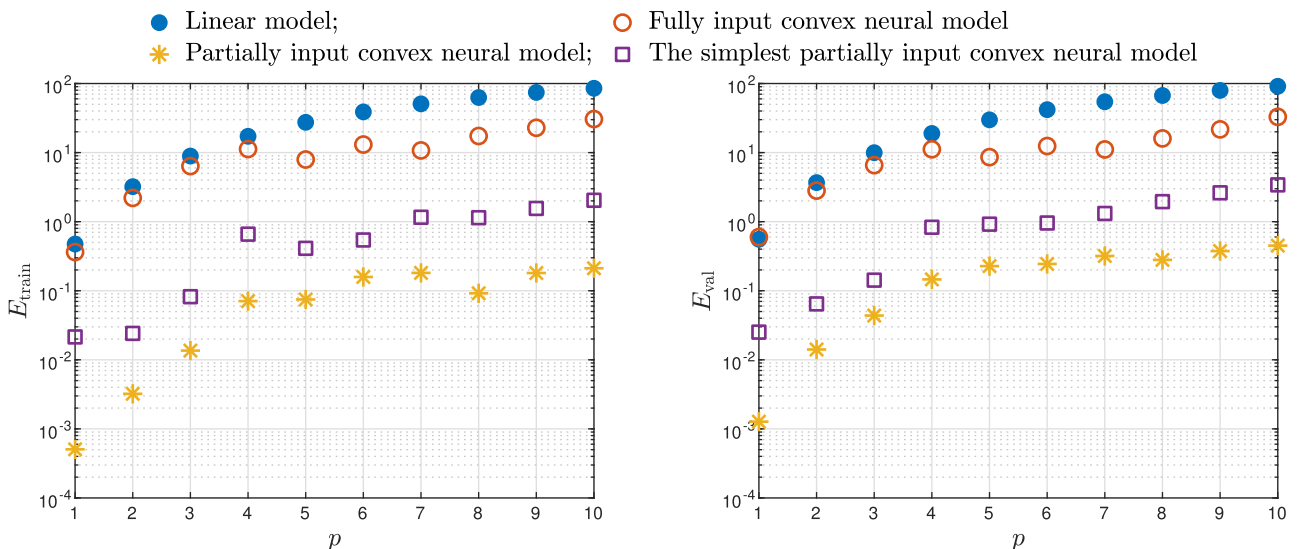


Fig. 15. Comparison of errors for training and validation data sets for four considered types of sub-models;  $p = 1, \dots, 10$ .

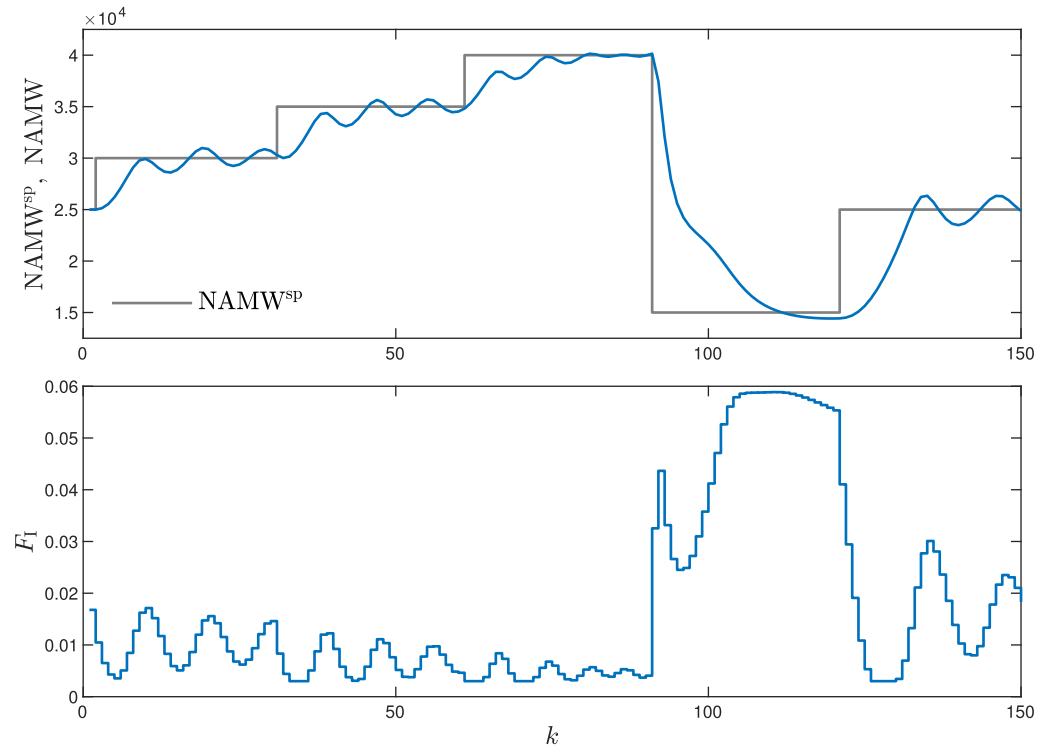


**Fig. 17.** Simulation results of the MPC algorithm using the linear model.

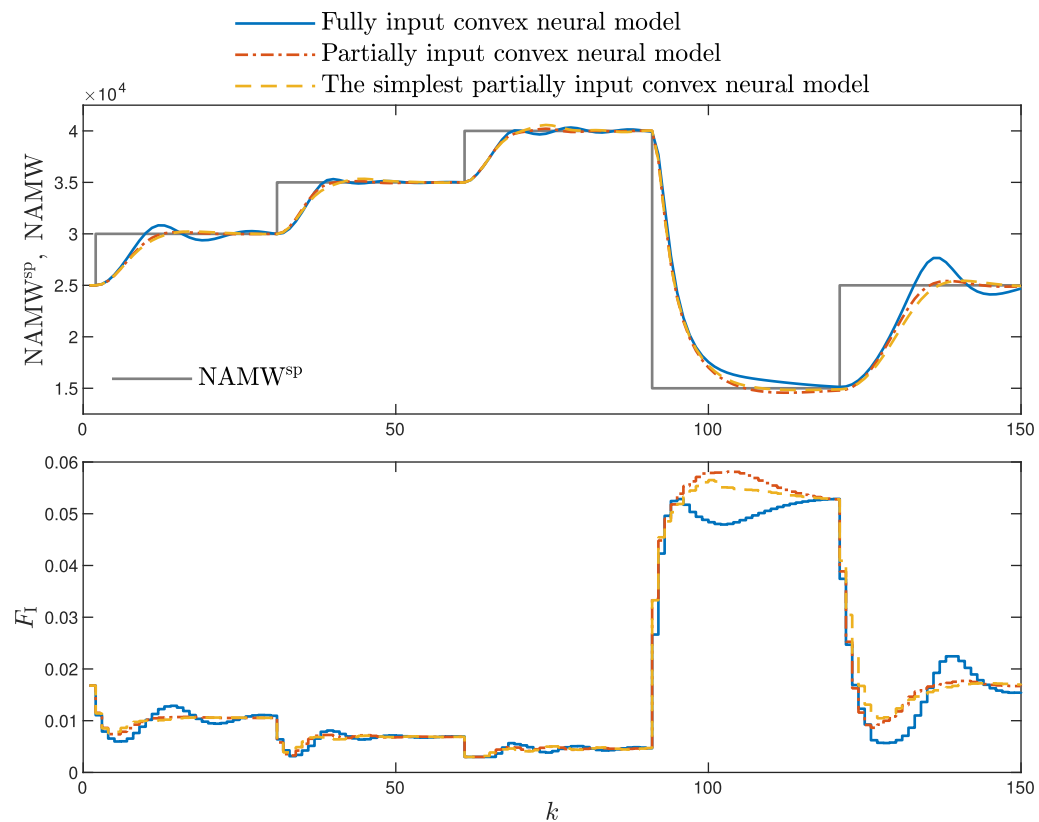


**Fig. 18.** Simulation results of MPC algorithms using three types of nonlinear models.

**Table 6**

Control quality indicators of MPC algorithms based on four types of considered models.

| Model                                            | $E_1$                | $E_2$                |
|--------------------------------------------------|----------------------|----------------------|
| Linear model                                     | $3.7875 \times 10^1$ | $3.2354 \times 10^1$ |
| Fully input convex neural model                  | $3.1143 \times 10^1$ | $2.7988 \times 10^1$ |
| Partially input convex neural model              | $2.8274 \times 10^1$ | $2.6992 \times 10^1$ |
| The simplest partially input convex neural model | $2.9643 \times 10^1$ | $2.7636 \times 10^1$ |

models is lower than that of the best ones, as depicted in Fig. 13, it is pretty good and the MPC algorithm compensates effectively for minor model inaccuracies (as a result of the negative feedback mechanism).

Having presented the obtained process trajectories, let us compare effectiveness of four considered models used in MPC in terms of two typical control quality indicators. Table 6 specifies the values of the sum of absolute values of control errors and the values of the sum of squared control errors (they are denoted by  $E_1$  and  $E_2$ , respectively); both indicators are measured for the whole simulation horizon. The obtained numerical indicators confirm observations when the trajectories presented in Figs. 17 and 18 are considered. Namely, the MPC algorithm based on the linear model gives the worse results; the fully input convex neural networks result in lower errors and the chosen partially input convex models yield the best results. It is essential that application of the simplest partially input convex models also gives quite good results, only slightly worse than in the case of the best partially input convex ones.

Let us stress that even the best partially input convex neural model is not a perfect one, i.e. it is characterised by low but non-zero errors. Despite that, the MPC algorithm can give zero steady-state error. To check robustness of MPC further, let us assume that the process is affected by an unmeasured disturbance

that acts on the process output. It changes its value: it is 0 for the sampling instants  $k < 15$ , it is 2000 for  $15 \leq k < 45$ , it is 1000 for  $45 \leq k < 75$ , it is  $-1000$  for  $75 \leq k < 105$ , it is 0 for  $k \geq 105$ . Fig. 19 shows simulation results of MPC when the linear model is used for prediction. Since accuracy of the linear model is low, the obtained control results are comparably poor to the disturbance-free case depicted in Fig. 17.

Finally, let us consider simulation results of the MPC algorithm based on three compared classes of neural models; they are presented in Fig. 20. When compared with the disturbance-free case shown in Fig. 18, we see the effect of changing disturbances at the sampling instants  $k = 15, 45, 75, 105$ . The controlled variable temporarily deviates from the required set-point, but the set-point is reached after several sampling instants thanks to the negative feedback mechanism. As far as control quality is concerned, we observe very similar process behaviour as in the disturbance-free case. Namely, the least accurate fully input convex neural model gives the slowest transient response for the fourth set-point change and large overshoot for the last step. Both partially input convex neural models give very good control quality, characterised by low overshoot and short regulation time. Control quality indicators are presented in Table 7. They confirm observations made considering the process trajectories. In particular, it is important that the partially input convex models obtain lower control errors than the fully convex ones and simplifying the partially input convex structure does not lead to any significant increase of the errors.

### 5.3.2. Comparison of calculation time

Let us study the influence of model architecture and horizons on the calculation time of MPC algorithms. The following eight types of partially input convex neural models are considered:

- the simplest model with one hidden layer and only one hidden node ( $K = 1$ ) as shown in Table 5,

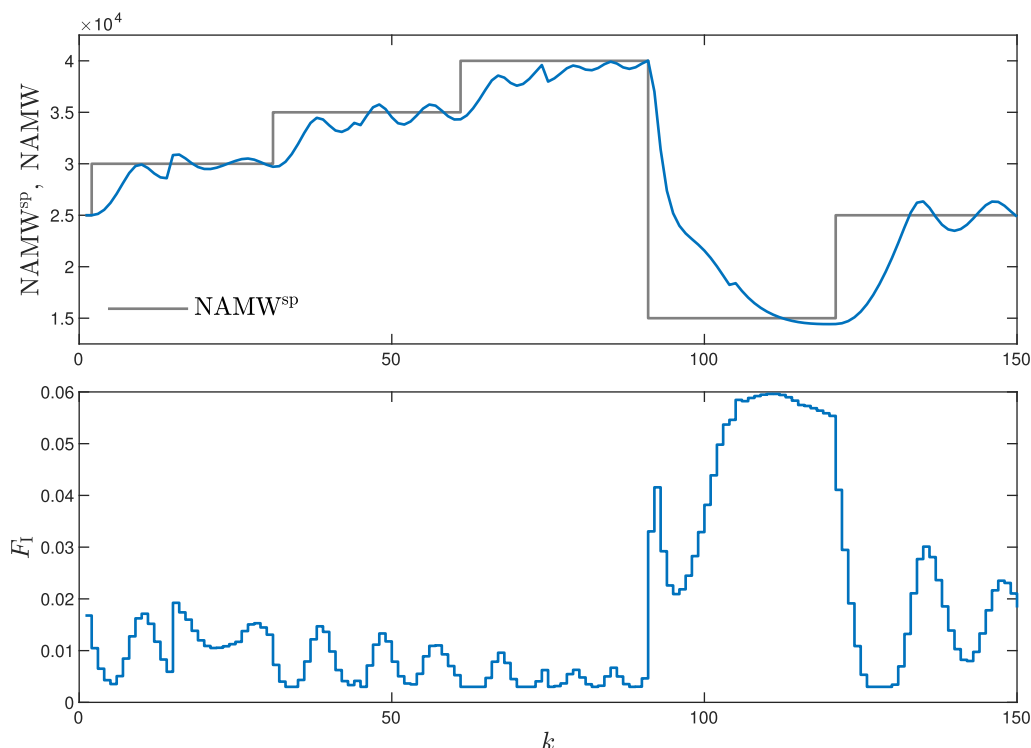


**Fig. 19.** Simulation results of the MPC algorithm using the linear model; the process is affected by unmeasured disturbances.

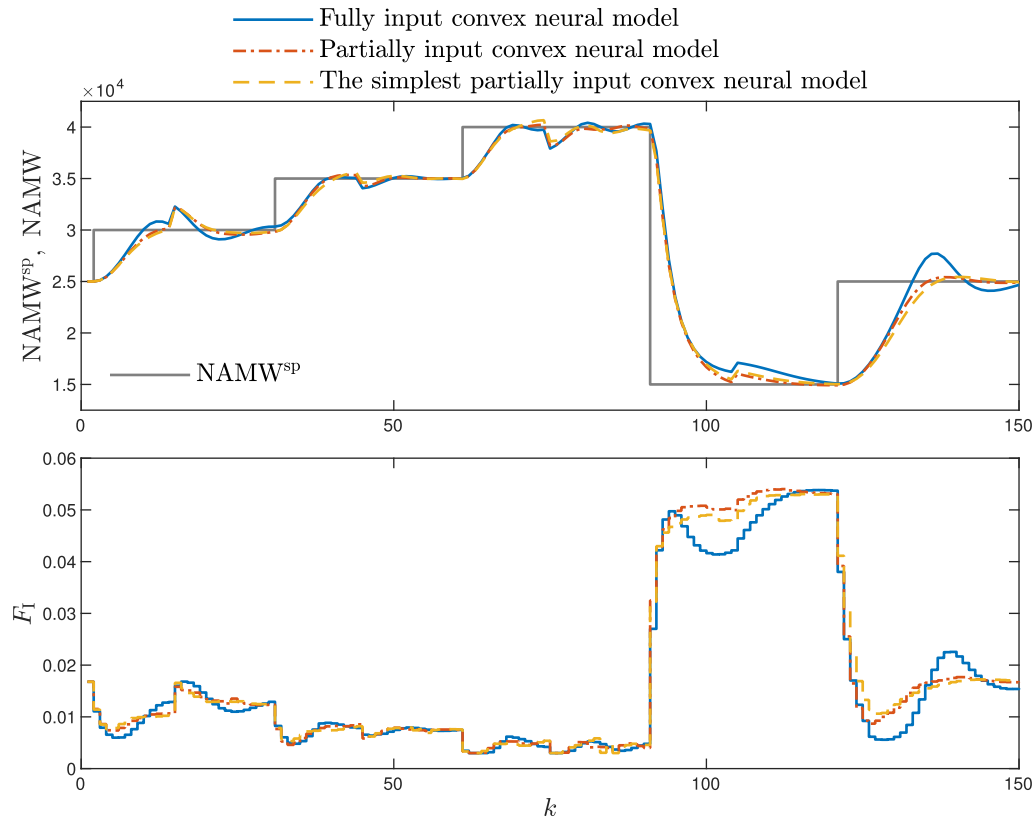


**Fig. 20.** Simulation results of MPC algorithms using three types of nonlinear models; the process is affected by unmeasured disturbances.

- b) the model with one hidden layer and the best selected number of hidden nodes as shown in Table 4,
- c) the model with one hidden layer and  $K = 10$  hidden nodes,
- d) the model with one hidden layer and  $K = 20$  hidden nodes,
- e) the model with one hidden layer and  $K = 40$  hidden nodes,
- f) the model with two hidden layers and  $K = [10 \ 10]$  hidden nodes,
- g) the model with three hidden layers and  $K = [10 \ 10 \ 10]$  hidden nodes,
- h) the model with four hidden layers and  $K = [10 \ 10 \ 10 \ 10]$  hidden nodes.

All models use the LeakyReLU activation function. Of course, for the considered process, it is sufficient to use the model with only one hidden layer and the optimised number of nodes listed in Table 4 because an increase in the number of nodes leads to worse accuracy, as shown in Fig. 11. Similarly, two, three or four hidden layers do not improve accuracy; they are only considered to study the calculation time. Two scenarios of horizons used in MPC are considered: in the first case,  $N = 5$  and  $N_u = 1, \dots, 5$ ; in the second case,  $N = 10$  and  $N_u = 1, \dots, 10$  (the control horizon cannot be

longer than the prediction one). Fig. 21 depicts the comparison of the calculation time for eight model variants mentioned above and two scenarios of horizons. The calculation time is scaled in such a way that the time for the default horizons ( $N = 10, N_u = 3$ ) and the network with one hidden layer containing the selected numbers of nodes corresponds to 100%. The following observations are possible:

1. The control horizon,  $N_u$ , has the most predominant influence on the calculation time. It is obvious since the control horizon defines the number of decision variables of the MPC optimisation task (3). The prediction horizon,  $N$ , also has some influence on the calculation time, but it is much weaker. It is important that the calculation time is a linear function of the control horizon.
2. As far as the relation between the model structure and the calculation time is concerned, in the case of the model with one hidden layer, the increase of the number of hidden nodes,  $K$ , results in lengthening the calculation time, but this influence is not very strong. The difference is clearly observed when we compare the calculation time for the simplest model with only one hidden node (Fig. 21a) and the model with the selected number of nodes (Fig. 21b). Further increase of the number of nodes results in a very limited lengthening of the calculation time as depicted in Fig. 21c (the network with  $K = 10$  nodes), Fig. 21d (the network with  $K = 20$  nodes) and Fig. 21e (the network with  $K = 40$  nodes). It is worth noting that it is possible because the network is implemented in MATLAB using a very efficient vector–matrix representation.
3. The calculation time increases significantly when the number of hidden layers is increased to two (Fig. 21f), three (Fig. 21g) and four (Fig. 21h).

**Table 7**  
Control quality indicators of MPC algorithm based on four types of considered models; the process is affected by unmeasured disturbances.

| Model                                            | $E_1$                | $E_2$                |
|--------------------------------------------------|----------------------|----------------------|
| Linear model                                     | $3.8920 \times 10^1$ | $3.1992 \times 10^1$ |
| Fully input convex neural model                  | $3.3030 \times 10^1$ | $2.7818 \times 10^1$ |
| Partially input convex neural model              | $2.9930 \times 10^1$ | $2.6544 \times 10^1$ |
| The simplest partially input convex neural model | $3.1403 \times 10^1$ | $2.6980 \times 10^1$ |

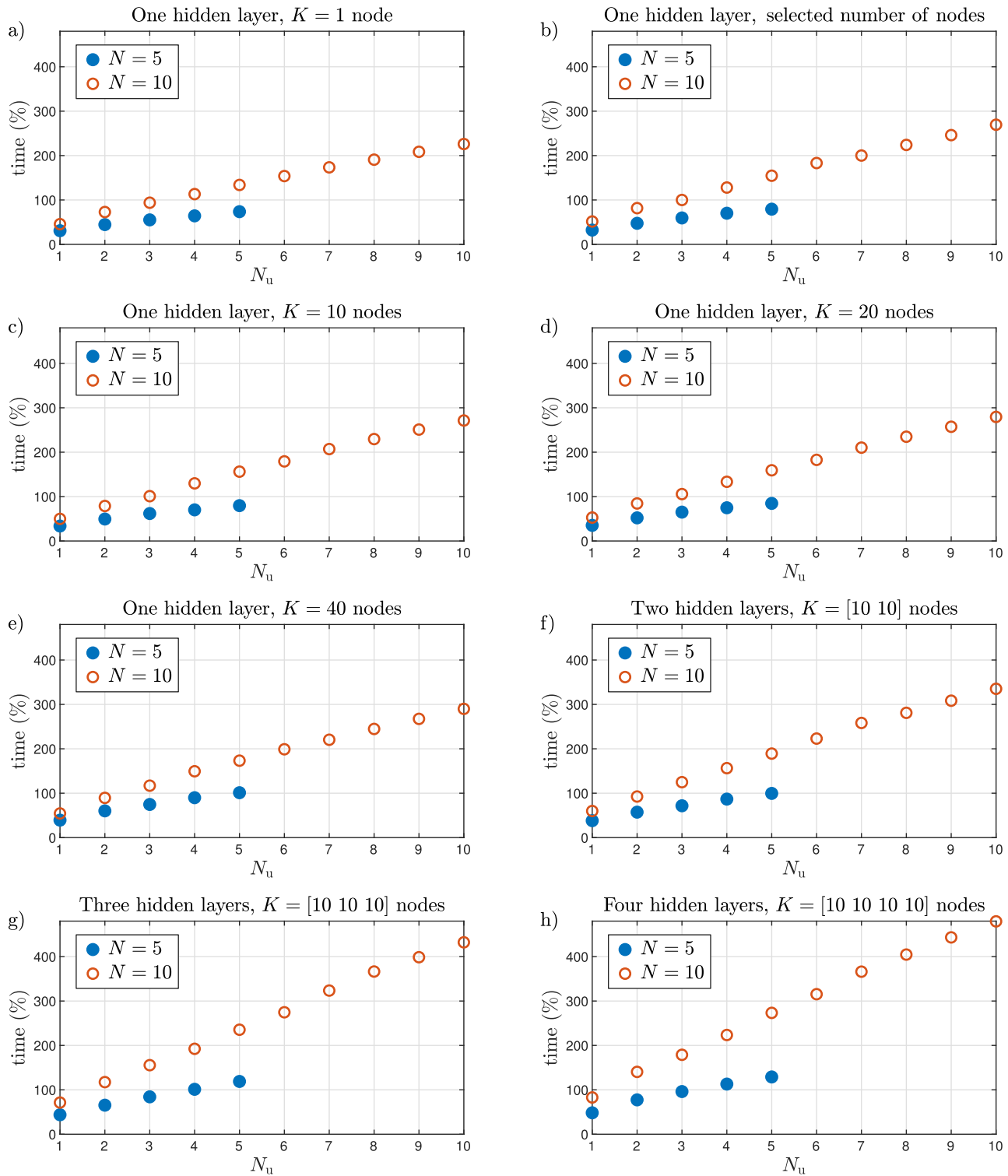


**Fig. 21.** Comparison of the calculation time of MPC algorithms using eight variants of the partially input convex neural multi-model and two scenarios of horizons.

Taking into account the reported calculated times, short control horizons are recommended, while the prediction horizon should be long enough to obtain good control quality. Moreover, we should avoid networks with multiple hidden layers and try to minimise the number of hidden nodes. One hidden layer turns out to be sufficient for the considered benchmark process.

In addition to the prediction and control horizon, the number of layers and the number of hidden nodes, the number of model inputs (defined by the constants  $n_A$  and  $n_B$ ) also influences the calculation time. Similarly, the type of the activation function used in the network also influences the calculation time. Naturally, the ReLU and LeakyReLU functions are calculated much

faster than the Softplus and LogSumExp ones, as discussed in Section 3.3.

## 6. Conclusions

This work presents the nonlinear MPC algorithm whose unique feature is convexity of the optimisation task solved online at each sampling instant. Such an advantage is possible because convex neural multi-models are used for prediction. Two classes of convex neural networks are considered: the fully input convex structure and the partially input convex one. As a result of using the multi-model approach, all sub-models are trained in the simple one-step-ahead non-recurrent mode and not used recurrently for prediction in MPC.

Effectiveness of the described multi-modelling is evaluated for a simulated polymerisation benchmark. The rudimentary fully input convex model turns out to be too restrictive to approximate precisely the process, particularly at the end of the prediction horizon. The partially input convex model with fewer parameters results in much better accuracy. It is also interesting that a simplified partially input convex model in which all sub-models have only one hidden node gives better results than the complex full input convex structure.

Next, the considered convex multi-models are used in MPC of the polymerisation benchmark; both disturbance-free and disturbance scenarios are considered. The MPC algorithm based on the partially input convex models gives excellent results in these two cases. In contrast, utilisation of the significantly less precise fully input convex neural multi-model gives much worse control quality.

Computational effectiveness resulting from convexity of the MPC optimisation task and the easiness of model development are significant advantages of the presented approach. However, two limitations must be named. Firstly, developing numerous sub-models may be an essential difficulty when the prediction horizon is long. It is obvious that the prediction is very good for time steps close to the current sampling instant, but the prediction accuracy is likely to be poor for very distant time steps. Hence, the presented solution is recommended for short and moderate prediction horizons. Secondly, the presented modelling and control methods are adequate when the considered process may be described by a nonlinear representation that is approximately convex in the future sequence of the manipulated variable. The convex models are not applicable when the process itself has a significantly non-convex nature.

Extension of the presented idea to multivariable processes is also possible. In such a case, each sub-model calculates predictions of all controlled variables for one future sampling instant. It is also possible to use an adaptive version of the MPC algorithm presented in this work when significant changes of process behaviour are expected. In such a case, the model is retrained on-line using successively collected data. The multi-model prediction scheme is the same in non-adaptive and adaptive realisations.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

- [1] B. Amos, L. Xu, and Z.J. Kolter. Input convex neural networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, volume 70 of Proceedings of Machine Learning Research, pages 146–155. PMLR, 2017.
- [2] S. Bhartiya, J.R. Whiteley. Factorized approach to nonlinear MPC using a radial basis function model, *AIChE J.* 47 (2001) 358–368.
- [3] A.D. Bonzanini, J.A. Paulson, G. Makrygiorgos, A. Mesbah, Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks, *Comput. Chem. Eng.* 145 (2021) 107174.
- [4] F. Bünnig, A. Schalbeter, A. Aboudonia, M. Hudoba de Badyn, P. Heer, J. Lygeros, Input convex neural networks for building MPC, *Proc. Mach. Learn. Res.* 144 (2021) 1–12.
- [5] G.C. Calafiore, S. Gaubert, C. Possieri, A universal approximation result for difference of Log-Sum-Exp neural networks, *IEEE Trans. Neural Networks Learn. Syst.* 31 (2020) 5603–5612.
- [6] E.F. Camacho, C. Bordons, *Model Predictive Control*, Springer, London, 1999.
- [7] Y. Chen, Y. Shi, B. Zhang, Optimal control via neural networks: a convex approach, in: *Proceedings of the International Conference on Learning Representations*, New Orleans, USA, 2019.
- [8] Y. Chen, Y. Shi, B. Zhang, Data-driven optimal voltage regulation using input convex neural networks, *Electric Power Syst. Res.* 189 (2020) 106741.
- [9] C. Cheng, M.-S. Chiu, Adaptive IMC controller design for nonlinear process control, *Chem. Eng. Res. Des.* 85 (2007) 234–244.
- [10] D.T. Doncevic, A.M. Schweidtmann, Y. Vaupel, P. Schäfer, A. Caspari, A. Mitsos, Deterministic global nonlinear model predictive control with recurrent neural networks embedded, *IFAC-PapersOnLine* 53 (2020) 5273–5278.
- [11] F.J. Doyle, B.A. Ogunnaike, R.K. Pearson, Nonlinear model-based control using second-order volterra models, *Automatica* 31 (1995) 697–714.
- [12] A. Grancharova, T.A. Johansen, *Explicit Nonlinear Model Predictive Control*, *Lecture Notes in Control and Information Sciences*, vol. 429, Springer, Berlin, 2012.
- [13] J.K. Gruber, D.R. Ramirez, D. Limon, T. Alamo, A convex approach for NMPC based on second order Volterra series model, *Robust Nonlinear Control* 25 (2015) 3546–3571.
- [14] T.A. Johansen, Approximate explicit receding horizon control of constrained nonlinear systems, *Automatica* 40 (2004) 293–300.
- [15] Y. Kansa, C. Min-Sen, Adaptive generalized predictive control based on JITL technique, *J. Process Control* 19 (2009) 1067–1072.
- [16] D.K. Klein, M. Fernández, R.J. Martin, P. Neff, O. Weeger, Polyconvex anisotropic hyperelasticity with neural networks, *J. Mech. Phys. Solids* 159 (2022) 104703.
- [17] M. Korda, I. Mezić, Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control, *Automatica* 93 (2018) 149–160.
- [18] M. Ławryńczuk, Efficient nonlinear predictive control based on structured neural models, *Int. J. Appl. Math. Comput. Sci.* 19 (2009) 233–246.
- [19] M. Ławryńczuk, Dynamic matrix control algorithm based on interpolated step response neural models. In L. Rutkowski, R. Scherer, R. Tadeusiewicz, L.A. Zadeh, and J.M. Zurada, editors, *Proceedings of the 9th International Conference on Artificial Intelligence and Soft Computing (ICAISC 2010)*, *Lecture Notes in Artificial Intelligence*, vol. 6114, pages 297–304, Zakopane, Poland, 2010.
- [20] M. Ławryńczuk, Training of neural models for predictive control, *Neurocomputing* 73 (2010) 1332–1343.
- [21] M. Ławryńczuk, *Computationally Efficient Model Predictive Control Algorithms: A Neural Network Approach*, *Studies in Systems, Decision and Control*, vol. 3, Springer, Cham, 2014.
- [22] M. Ławryńczuk, Explicit nonlinear predictive control algorithms with neural approximation, *Neurocomputing* 129 (2014) 570–584.
- [23] M. Ławryńczuk, Nonlinear state-space predictive control with on-line linearisation and state estimation, *Int. J. Appl. Math. Comput. Sci.* 25 (2015) 833–847.
- [24] M. Ławryńczuk, Modelling and predictive control of a neutralisation reactor using sparse Support Vector Machine Wiener models, *Neurocomputing* 205 (2016) 311–328.
- [25] M. Ławryńczuk, Nonlinear Predictive Control Using Wiener Models: Computationally Efficient Approaches for Polynomial and Neural Structures, *Studies in Systems, Decision and Control*, vol. 389, Springer, Cham, 2022.
- [26] S. Li, Y. Li, Model predictive control of an intensified continuous reactor using a neural network Wiener model, *Neurocomputing* 185 (2016) 93–104.
- [27] P.F. Lima, G.C. Pereira, J. Mårtensson, B. Wahlberg, Experimental validation of model predictive control stability for autonomous driving, *Control Eng. Practice* 81 (2018) 244–255.
- [28] J.M. Maciejowski, *Predictive control with constraints*, Prentice Hall, Harlow, 2002.
- [29] J. Madár, J. Abonyi, F. Szeifert, Genetic programming for the identification of nonlinear input-output models, *Ind. Eng. Chem. Res.* 44 (2005) 3178–3186.
- [30] E.T. Maddalena, C.G. da S. Moraes, G. Waltrich, and C.N. Jones, A neural network architecture to learn explicit MPC controllers from data. *IFAC-PapersOnLine*, 53:11362–11367, 2020.
- [31] P.M. Marusak, A numerically efficient fuzzy MPC algorithm with fast generation of the control signal, *Int. J. Appl. Math. Comput. Sci.* 31 (2021) 59–71.
- [32] R. Moriyasu, T. Ikeda, S. Kawaguchi, K. Kashima, Structured Hammerstein-Wiener model learning for model predictive control, *IEEE Control Syst. Lett.* 6 (2022) 397–402.
- [33] M. Mulas, S. Tronci, F. Corona, H. Haimi, P. Lindell, M. Heinonen, R. Vahala, R. Baratti, Predictive control of an activated sludge process: An application to the Viikinki wastewater treatment plant, *Control Eng. Practice* 35 (2015) 89–100.

- [34] S. Ogonowski, D. Bismor, Z. Ogonowski, Control of complex dynamic nonlinear loading process for electromagnetic mill, *Arch. Control Sci.* 30 (2020) 471–500.
- [35] Z. Peng, C. Meng, L. Liu, D. Wang, T. Li, PWM-driven model predictive speed control for an unmanned surface vehicle with unknown propeller dynamics based on parameter identification and neural prediction, *Neurocomputing* 432 (2021) 1–9.
- [36] G. Ravi Srinivas, Y. Arkun, A global solution to the nonlinear model predictive control algorithms using polynomial ARX models, *Comput. Chem. Eng.* 21 (1997) 431–439.
- [37] G. Shafiee, M.M. Arefi, M.R. Jahed-Motlagh, A.A. Jalali, Nonlinear predictive control of a polymerization reactor based on piecewise linear Wiener model, *Chem. Eng. J.* 143 (2008) 282–292.
- [38] J. Shin, H. Jin Kim, S. Park, Y. Kim, Model predictive flight control using adaptive support vector regression, *Neurocomputing* 73 (2010) 1031–1037.
- [39] D.K. Siong Tok, Y. Shi, Y. Tian, D.-L. Yu, Factorized f-step radial basis function model for model predictive control, *Neurocomputing* 239 (2017) 102–112.
- [40] J. Sun, P. Li, C. Wang, Optimise transient control against DoS attacks on ESS by input convex neural networks in a game, *Sustain. Energy Grids Networks* 28 (2021) 100535.
- [41] P. Tatjewski, *Advanced control of industrial processes, structures and algorithms*, Springer, London, 2007.
- [42] P. Tatjewski, M. Ławryńczuk, Algorithms with state estimation in linear and nonlinear model predictive control, *Comput. Chem. Eng. Trans.* 143 (2020) 107065.
- [43] X. Wang, V. Mahalec, F. Qian, Globally optimal nonlinear model predictive control based on multi-parametric disaggregation, *J. Process Control* 52 (2017) 1–13.
- [44] X. Xia, D. Huang, Y. Jin, Nonlinear adaptive predictive control based on orthogonal wavelet networks, in: *Proceedings of the 4th World Congress on Intelligent Control and Automation*, Shanghai, China, 2002, pp. 305–311.
- [45] S. Yang, B.W. Bequette, Optimization-based control using input convex neural networks, *Comput. Chem. Eng.* 144 (2020) 107143.
- [46] T. Yang, W. Qiu, Y. Ma, M. Chadli, L. Zhang, Fuzzy model-based predictive control of dissolved oxygen in activated sludge processes, *Neurocomputing* 136 (2014) 88–95.
- [47] Y. Zheng, J. Zhou, W. Zhu, C. Zhang, C. Li, W. Fu, Design of a multi-mode intelligent model predictive control strategy for hydroelectric generating unit, *Neurocomputing* 207 (2016) 287–299.



**Maciej Ławryńczuk** was born in Warsaw, Poland, in 1972. He obtained his MSc in 1998, PhD in 2003 and DSc in 2013, in automatic control, from Warsaw University of Technology, Faculty of Electronics and Information Technology. Currently, he is employed at the same university at the Institute of Control and Computation Engineering as a professor (full). He is the author or a co-author of 7 books and more than 190 other publications, including more than 70 journal articles. His research interests include advanced control algorithms, particularly Model Predictive Control (MPC) algorithms, set-point optimisation algorithms, soft computing methods, particularly neural networks, modelling and simulation. Currently, Maciej Ławryńczuk is an associate editor of *ISA Transactions* (Elsevier), a member of the editorial board of *International Journal of Applied Mathematics and Computer Science* (De Gruyter), an academic editor of *Electronics* (MDPI) and *Computational Intelligence and Neuroscience* (Hindawi) journals.