Neural Networks in Model Predictive Control

Maciej Ławryńczuk

Institute of Control and Computation Engineering, Warsaw University of Technology
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
Tel.: +48 22 234-76-73
M.Lawrynczuk@ia.pw.edu.pl

Abstract. This work is concerned with Model Predictive Control (MPC) algorithms in which neural models are used on-line. Model structure selection, training and stability issues are thoroughly discussed. Computationally efficient algorithms are recommended which use on-line linearisation of the neural model and need solving on-line quadratic optimisation tasks. It is demonstrated that they give very good results, comparable to those obtained when nonlinear optimisation is used on-line in MPC. In order to illustrate the effectiveness of discussed approaches, a chemical process is considered. The development of appropriate models for MPC is discussed, the control accuracy and the computational complexity of recommended MPC are shown.

Keywords: Process control, Model Predictive Control, neural networks, model identification, optimisation, quadratic programming, linearisation.

1 Introduction

Model Predictive Control (MPC) is recognised as the only advanced control technique which has been very successful in practice [29, 43, 44, 47]. It is mainly because MPC algorithms can take into account constraints imposed on both process inputs (manipulated variables) and outputs (controlled variables), which usually decide on quality, economic efficiency and safety. Moreover, MPC techniques are very efficient in multivariable process control (i.e. for processes with many inputs and outputs) and for processes with difficult dynamic properties, e.g. with significant time-delays. Different versions of MPC algorithms are nowadays used in numerous fields, not only in chemical, food and motor industries, but also in medicine and aerospace [43].

MPC techniques based on easy to obtain linear models are frequently used in practice [43]. In many cases their control accuracy is sufficient, much better than that of the classical PID approach. Nevertheless, in the last two decades numerous MPC algorithms based on nonlinear models have been developed and have gained in popularity [12, 35, 43, 47, 48]. When applied to significantly nonlinear processes, they significantly improve the control accuracy in comparison with MPC approaches which use linear models.

In MPC a dynamic model of the process is used to predict its behaviour over some time horizon and to determine the optimal future control policy. Hence, the

N.T. Nguyen and E. Szczerbicki (Eds.): Intel. Sys. for Know. Management, SCI 252, pp. 31–63.
springerlink.com

choice of the model structure which is used in nonlinear MPC is extremely important. Main measures of model utility are: approximation accuracy, suitability for control and easiness of development [39]. Fundamental (first-principle) models [19,30], although potentially very precise, are usually not suitable for on-line control. Such models are comprised of systems of nonlinear differential and algebraic equations which have to be solved on-line in MPC at each sampling instant. It is usually computationally demanding as fundamental models can be very complex and may lead to numerical problems (e.g. stiffness, ill-conditioning). Moreover, in many cases the development and validation of fundamental models is difficult, it needs technological knowledge.

This work presents MPC algorithms based on neural models [11]. In spite of the fact that a number of different nonlinear black-box model types are available (e.g. polynomial models, fuzzy models, Volterra series models [39]), neural networks have some unique features thanks to which they can be very efficiently used on-line in MPC. More specifically, neural networks are universal approximators [13], have relatively a small number of parameters and a simple structure. Moreover, they directly describe relations between inputs and outputs of the process, which means that during on-line control it is not necessary to solve complicated systems of nonlinear differential equations at each sampling instant as it is necessary when fundamental models are used. In consequence, neural models can be effectively used in MPC as models of technological processes. In particular, thanks to a simple, regular structure of neural models, the implementation of described algorithms is relatively easy. Neural models are trained using recorded data sets, no technological knowledge is necessary.

Although the literature concerned with MPC algorithms based on neural models is quite rich [1,2,3,4,5,8,10,12,14,16,18,20,21,22,23,24,25,26,27,28,33,35,37,40,41,42,47,48,49,50,51,52], there are a few issues worth exploring. First of all, the way the nonlinear model is used on-line in MPC is crucial in light of the computational complexity and reliability of the whole control system. In theory, neural models can be used directly without any simplifications but it means that the optimal control policy at each sampling instant must be calculated from a nonlinear optimisation problem. It is not only computationally very demanding but also the optimisation routine is likely to terminate at shallow local minima. In order to reduce the computational burden and increase reliability, in this work suboptimal approaches are recommended. The nonlinear neural model is linearised on-line and next the obtained local approximation is used in MPC algorithms [10,20,21,22,23,24,25,26,27,28,37,47,48]. Thanks to using for control a local linearisation of the original neural model, the necessity of on-line nonlinear optimisation is avoided, it is replaced by an easy to solve quadratic programming problem. It is demonstrated in this work that suboptimal MPC algorithms with on-line linearisation are very precise, the control accuracy is comparable to that obtained when a nonlinear optimisation routine is used on-line at each sampling instant.

The second important issue discussed in this work is the choice of the model structure and training. It is emphasised that MPC algorithms are very

model-based, the possible control performance is determined by the accuracy of predictions calculated by means of a dynamic model. The role of the model in MPC cannot be ignored during model structure selection and training. The model has to be able to make good predictions of future behaviour of the process over the whole prediction horizon. Usually, Multi Layer Perceptron (MLP) and Radial Basis Function (RBF) neural networks are used in MPC. In some cases block-oriented nonlinear models which are composed of linear dynamic systems and nonlinear steady-state (static) elements can be efficiently used (neural Hammerstein and Wiener models). In this work two classes of specialised neural models designed with the specific aim of using them in MPC are also discussed: multi-models [24] and structured models [20]. Both specialised models are trained easily as one-step ahead predictors, but they calculate predictions for the whole prediction horizon without being used recurrently. As a result, the prediction error is not propagated which is particularly important in practice.

In spite of the fact that in practice MPC techniques are relatively easily tuned, a stable version of described algorithms based on different kinds of neural models is shortly discussed.

In order to show the effectiveness of discussed approaches to MPC a chemical reactor process is discussed. The development of appropriate models for MPC is thoroughly discussed, the control accuracy and the computational complexity of recommended MPC are also shown.

2 Model Predictive Control Problem Formulation

In MPC algorithms [29, 44, 47] at each consecutive sampling instant k a set of future control increments is calculated

$$\triangle \boldsymbol{u}(k) = \begin{bmatrix} \triangle u(k|k) \\ \vdots \\ \triangle u(k+N_u-1|k) \end{bmatrix} \tag{1}$$

It is assumed that $\triangle u(k+p|k) = 0$ for $p \geq N_u$, where N_u is the control horizon. The objective is to minimise differences between the reference trajectory $y^{\mathrm{ref}}(k+p|k)$ and predicted output values $\hat{y}(k+p|k)$ over the prediction horizon $N \geq N_u$ and to penalise excessive control increments. The minimised cost function is usually

$$J(k) = \sum_{p=1}^{N} (y^{\mathrm{ref}}(k+p|k) - \hat{y}(k+p|k))^2 + \sum_{p=0}^{N_u-1} \lambda_p (\triangle u(k+p|k))^2 \tag{2}$$

where $\lambda_p > 0$ are weighting coefficients. Only the first element of the determined sequence (1) is applied to the process, i.e. $u(k) = \triangle u(k|k) + u(k-1)$. At the next sampling instant, $k+1$, the prediction is shifted one step forward and the whole procedure is repeated. Fig. 1 illustrates the general idea of MPC.

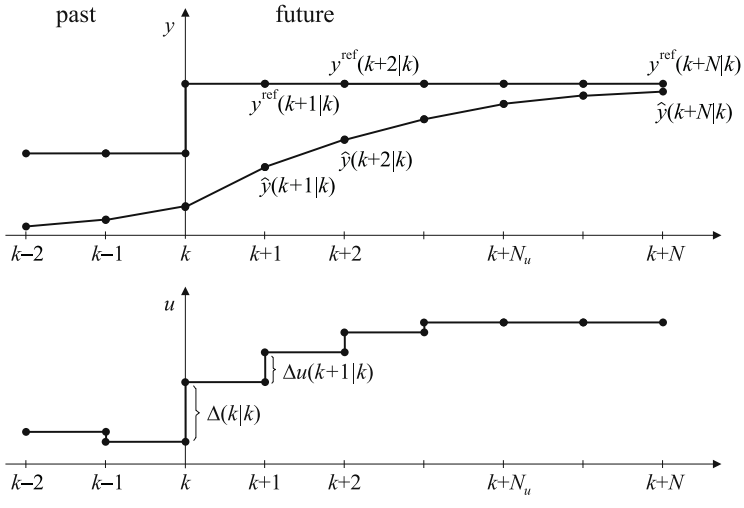

Fig. 1. The principle of predictive control

A unique feature of MPC is the fact that constraints imposed on process variables can be rigorously taken into account. More specifically, future control increments (1) are found on-line from the following optimisation problem

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

where $u^{\min}$, $u^{\max}$, $\triangle u^{\max}$, $y^{\min}$, $y^{\max}$ define constraints imposed on the magnitude of the input variable, the increment of the input variable and the magnitude of the output variable, respectively.

MPC algorithms directly use an explicit dynamic model in order to predict future behaviour of the process, i.e. to calculate predicted values of the output variable, $\hat{y}(k+p|k)$, over the prediction horizon ($p=1,\ldots,N$). Hence, the role of the model in MPC is crucial. As a result, MPC techniques are very model-based, the accuracy of the model significantly affects the quality of control.

A great advantage of MPC algorithms is the fact that they can be efficiently used for multivariable processes. Assuming that the process has n_u inputs and n_y outputs (i.e. $u(k) \in \Re^{n_u}$, $y(k) \in \Re^{n_y}$), the MPC cost function (2) becomes

$$J(k) = \sum_{p=1}^{N} \left\| y^{\text{ref}}(k+p|k) - \hat{y}(k+p|k) \right\|^2_{\boldsymbol{M}_p} + \sum_{p=0}^{N_u-1} \left\| \triangle u(k+p|k) \right\|^2_{\boldsymbol{\Lambda}_p} \tag{4}$$

where $\|x\|^2_{\boldsymbol{A}} = x^T \boldsymbol{A} x$, $\boldsymbol{M}_p \geq 0$ and $\boldsymbol{\Lambda}_p > 0$ are weighting matrices of dimensionality $n_y \times n_y$ and $n_u \times n_u$, respectively. Having completed the MPC optimisation task (3), the first n_u elements of the sequence (1) are applied to the process.

3 Neural Models of the Process

Let the dynamic process under consideration be described by the following discrete-time Nonlinear Auto Regressive with eXternal input (NARX) model

$$y(k) = f(\boldsymbol{x}(k)) = f(u(k-\tau), \ldots, u(k-n_B), y(k-1), \ldots, y(k-n_A)) \tag{5}$$

where $f : \Re^{n_A+n_B-\tau+1} \longrightarrow \Re$ is a nonlinear function which describes the model, integers n_A, n_B, τ define the order of dynamics, $\tau \leq n_B$. Although, in general, the function f can be realised by various nonlinear models [39], Multi Layer Perceptron (MLP) and Radial Basis Functions (RBF) neural networks are most frequently used. Both structures are universal approximators [13] capable of approximating any smooth function to an arbitrary degree of accuracy, they have relatively a small number of parameters and a simple structure.

3.1 MLP Neural Model

When the MLP neural network with one hidden layer and a linear output [11] is used as the function f in (5), the output of the model can be expressed as

$$y(k) = f(\boldsymbol{x}(k)) = w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi(z_i(k)) \tag{6}$$

where $z_i(k)$ is a sum of inputs of the i^{th} hidden node, $\varphi : \Re \longrightarrow \Re$ is the nonlinear transfer function (e.g. hyperbolic tangent), K is the number of hidden nodes. From (5) one has

$$z_i(k) = w_{i,0}^1 + \sum_{j=1}^{I_u} w_{i,j}^1 u(k-\tau+1-j) + \sum_{j=1}^{n_A} w_{i,I_u+j}^1 y(k-j) \tag{7}$$

Weights of the network are denoted by $w_{i,j}^1$, $i = 1, \ldots, K$, $j = 0, \ldots, n_A + n_B - \tau + 1$, and w_i^2, $i = 0, \ldots, K$, for the first and the second layer, respectively, $I_u = n_B - \tau + 1$.

In the case of the multivariable processes with n_u inputs and n_y outputs the model is usually comprised of n_y independent Multiple-Input Single-Output (MISO) models. The whole nonlinear model has the general form

$$\begin{aligned} y_1(k) = f_1(&u_1(k-\tau^{1,1}), \ldots, u_1(k-n_B^{1,1}), \ldots, \\ &u_{n_u}(k-\tau^{1,n_u}), \ldots, u_{n_u}(k-n_B^{1,n_u}), y_1(k-1), \ldots, y_1(k-n_A^1)) \\ &\vdots \end{aligned} \tag{8}$$

$$\begin{aligned} y_{n_y}(k) = f_{n_y}(&u_1(k-\tau^{n_y,1}), \ldots, u_1(k-n_B^{n_y,1}), \ldots, \\ &u_{n_u}(k-\tau^{n_y,n_u}), \ldots, u_{n_u}(k-n_B^{n_y,n_u}), y_{n_y}(k-1), \ldots, y_{n_y}(k-n_A^{n_y})) \end{aligned} \tag{9}$$

where integers n_A^m, $n_B^{m,n}$, $\tau^{m,n}$ for $m = 1, \ldots, n_y$, $n = 1, \ldots, n_u$ define the order of dynamics, $\tau^{m,n} \leq n_B^{m,n}$. Functions $f_m : \Re^{n_A^m + \sum_{n=1}^{n_u}(n_B^{m,n} - \tau^{m,n}+1)} \longrightarrow \Re$, $m = 1, \ldots, n_y$ are realised by independent neural networks which are trained separately. Alternatively, the model can be realised by only one neural network with as many as n_y outputs, but training of such models is usually more difficult.

Unlike fundamental models, which are comprised of systems of nonlinear algebraic and differential equations, MLP and RBF neural models directly describe relations between inputs and outputs of the process. Thanks to it, neural models can be effectively used in MPC since during on-line calculation of the control policy it is not necessary to solve these equations at each sampling instant, which may be computationally complex and lead to numerical problems (e.g. stiffness, ill-conditioning).

3.2 RBF Neural Model

If the RBF neural network containing one hidden layer with Gaussian functions and a linear output is used as the function f in (5), the output of the model is

$$\begin{aligned} y(k) = f(\boldsymbol{x}(k)) =& w_0 + \sum_{i=1}^{K} w_i \exp(-\|\boldsymbol{x}(k) - \boldsymbol{c}_i\|^2_{\boldsymbol{Q}_i}) \\ =& w_0 + \sum_{i=1}^{K} w_i \exp(-z_i(k)) \end{aligned} \tag{10}$$

where K is the number of hidden nodes. Vectors $\boldsymbol{c}_i \in \Re^{n_A+n_B-\tau+1}$ and the diagonal weighting matrices $\boldsymbol{Q}_i = diag(q_{i,1}, \ldots, q_{i,n_A+n_B-\tau+1})$ describe centres and widths of nodes, respectively, $i = 1, \ldots, K$. The model (10) is sometimes named Hyper Radial Basis Function (HRBF) neural network in contrast to the ordinary RBF neural networks in which widths of nodes are constant. Let $z_i(k)$ be the sum of inputs of the i-th hidden node. Recalling the arguments of the rudimentary model (5), one has

$$z_i(k) = \sum_{j=1}^{I_u} q_{i,j}(u(k-\tau+1-j) - c_{i,j})^2 + \sum_{j=1}^{n_A} q_{i,I_u+j}(y(k-j) - c_{i,I_u+j})^2 \tag{11}$$

Although it is a well known fact that both MLP and RBF neural models are universal approximators, MLP networks are global ones whereas RBF networks are local ones. It is because in the first case all hidden nodes are used to calculate the output for a given input, in the second case only selected hidden nodes are employed, other nodes are practically inactive. A direct consequence of this fact is that for MLP networks it is difficult to establish a link between available data and parameters of hidden nodes. As a result, weights of such networks are usually initialised randomly, training should be repeated many times for different numbers of hidden nodes to find the adequate topology which gives good approximation. Conversely, training of RBF models is much more efficient because parameters of basis functions are directly found from available data. Moreover,

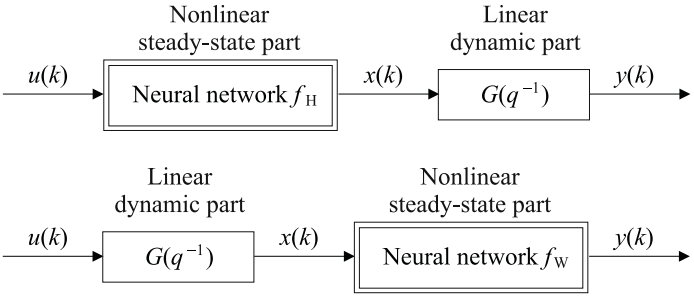

Fig. 2. The structure of neural Hammerstein (*top*) and Wiener (*bottom*) models

selection of the optimal structure of the RBF model which leads to desired approximation accuracy can be included in a training procedure, whereas selection of the structure of the MLP model usually needs training many networks.

3.3 Hammerstein and Wiener Neural Models

Neural MLP and RBF models are entirely black-box models. It means that the model structure has nothing to do with the physical nature of the process and model parameters (weights) have no physical interpretation. An interesting alternative is to use block-oriented nonlinear models which are composed of linear dynamic systems and nonlinear steady-state (static) elements. Hammerstein and Wiener models are most known and most widely implemented members of this class [15]. A model is called the Hammerstein model if the linear dynamic part follows the nonlinear steady-state one, in the Wiener model the connection order is reversed as shown in Fig. 2. Unlike black-box models, block-oriented models have a clear interpretation, the steady-state part describes the gain of the system.

In the simplest case polynomials can be used as the steady-state nonlinear part of block-oriented models. Unfortunately, some nonlinear functions need polynomials of a high order. In such cases models are complex and the model uncertainty is likely to be increased. Moreover, polynomials are likely to have oscillatory interpolation and extrapolation properties. In consequence, the application of polynomials is in practice limited [15].

A sound alternative is to use neural networks in the steady-state part of Hammerstein and Wiener models. Such an approach has a few advantages. Not only are neural networks universal approximators, but also, in contrast to polynomial approximators, neural approximations are very smooth, they do not suffer from oscillatory interpolation and extrapolation behavior. An excellent review of identification algorithms and applications of block-oriented Hammerstein and Wiener models is given in [15].

Both MLP and RBF neural networks can be used as the nonlinear steady-state part of block-oriented models. When the MLP neural network is used, the nonlinear steady-state function f_H of the Hammerstein model is described by

$$x(k) = f_{\mathrm{H}}(u(k)) = w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi(w_{i,0}^1 + w_{i,1}^1 u(k)) \tag{12}$$

where $x(k)$ is an auxiliary signal. Weights are denoted by $w_{i,j}^1$, $i = 1, \ldots, K$, $j = 0, 1$ and w_i^2, $i = 0, \ldots, K$, for the first and the second layer, respectively.

The transfer function of the linear dynamic part is

$$G(q^{-1}) = \frac{\boldsymbol{B}(q^{-1})}{\boldsymbol{A}(q^{-1})} = \frac{b_\tau q^{-\tau} + \ldots + b_{n_B} q^{-n_B}}{1 + a_1 q^{-1} + \ldots + a_{n_A} q^{-n_A}} \tag{13}$$

where q^{-1} is the backward shift operator. Hence, the output of the dynamic part is

$$y(k) = \sum_{l=\tau}^{n_B} b_l x(k-l) - \sum_{l=1}^{n_A} a_l y(k-l) \tag{14}$$

Combining (12) and (14), the output of the neural Hammerstein model is

$$y(k) = \sum_{l=\tau}^{n_B} b_l \left(w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi(w_{i,0}^1 + w_{i,1}^1 u(k-l)) \right) - \sum_{l=1}^{n_A} a_l y(k-l) \tag{15}$$

Analogously, for the neural Wiener model one has

$$\begin{aligned} y(k) = f_{\mathrm{W}}(x(k)) =& w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi(w_{i,0}^1 + w_{i,1}^1 x(k)) \\ =& w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi\left(w_{i,0}^1 + w_{i,1}^1 \left(\sum_{l=\tau}^{n_B} b_l u(k-l) - \sum_{l=1}^{n_A} a_l x(k-l) \right) \right) \end{aligned} \tag{16}$$

Similarly as rudimentary MLP and RBF neural models, block-oriented neural models have a regular structure, unlike fundamental models they do not contain differential and algebraic equations which have to be solved on-line in MPC at each sampling instant. Hence, they can be also easily used in MPC.

4 MPC with Nonlinear Optimisation (MPC-NO)

In MPC algorithms a nonlinear model is directly used to calculate predictions $\hat{y}(k+p|k)$ which are taken into account in the minimised cost function (2). The general prediction equation is

$$\hat{y}(k+p|k) = y(k+p|k) + d(k) \tag{17}$$

where quantities $y(k+p|k)$ are calculated from the model of the process. Rudimentary MLP (6), (7), RBF (10), (11) neural models or block oriented neural Hammerstein (15) and Wiener (16) models can be used. Usually, the "DMC

type" disturbance model is used, in which the unmeasured disturbance $d(k)$ is assumed to be constant over the prediction horizon [47]. It is estimated from

$$d(k) = y(k) - y(k|k-1) \tag{18}$$

where $y(k)$ is measured while $y(k|k-1)$ is calculated from the model.

If for prediction in MPC a nonlinear model is used without any simplifications, predictions $\hat{y}(k+p|k)$ depend in a nonlinear way on the calculated control policy, i.e. on future control increments $\triangle \boldsymbol{u}(k)$. It means that the MPC optimisation problem (3) becomes a nonlinear task which has to be solved on-line in real time. It may be computationally demanding and time consuming. Moreover, such an approach may be not reliable in practice because the nonlinear optimisation routine is likely to terminate in a shallow local minimum. Nevertheless, simulation results of the MPC-NO algorithm based on different neural models are frequently presented in the literature, its apparent computational inefficiency and limited practical applicability are overlooked [2, 3, 14, 16, 33, 37, 49, 50, 51, 52].

5 MPC with Nonlinear Prediction and Linearisation (MPC-NPL)

In this work the MPC algorithm with Nonlinear Prediction and Linearisation (MPC-NPL) [26,47,48] is recommended. At each sampling instant k a local linear approximation of the nonlinear neural model is found on-line. Thanks to linearisation, predictions of the output depend in a purely linear way on the calculated control policy. As a result, the MPC-NPL algorithm needs solving on-line a quadratic programming problem, which can be easily completed within a foreseeable time period, the necessity of nonlinear optimisation is avoided. In practice, for different technological processes the algorithm gives good closed-loop control performance, comparable to that obtained in computationally demanding MPC-NO approach with full nonlinear optimisation [20,21,22,23,24,25,26,27,28,47,48].

The linear approximation of the nonlinear neural model (5) is

$$y(k) = \sum_{l=1}^{n_B} b_l(k)(u(k-l)) - \sum_{l=1}^{n_A} a_l(k)(y(k-l)) \tag{19}$$

where $a_l(k)$ and $b_l(k)$ are coefficients of the linearised model. Using the linearised model (19) recurrently, from the general prediction equation (17) one obtains

$$\begin{aligned} \hat{y}(k+1|k) =& b_1(k)u(k|k) + b_2(k)u(k-1) + b_3(k)u(k-2) + \ldots \\ &+ b_{n_B}(k)u(k-n_B+1) \\ &- a_1(k)y(k) - a_2(k)y(k-1) - a_3(k)y(k-2) - \ldots \\ &- a_{n_A}(k)y(k-n_A+1) + d(k) \end{aligned} \tag{20}$$

$$\begin{aligned}\hat{y}(k+2|k) =& b_1(k)u(k+1|k) + b_2(k)u(k|k) + b_3(k)u(k-1) + \ldots \\ &+ b_{n_B}(k)u(k-n_B+2) \\ &- a_1(k)\hat{y}(k+1|k) - a_2(k)y(k) - a_3(k)y(k-1) - \ldots \\ &- a_{n_A}(k)y(k-n_A+2) + d(k)\end{aligned} \tag{21}$$

$$\begin{aligned}\hat{y}(k+3|k) =& b_1(k)u(k+2|k) + b_2(k)u(k+1|k) + b_3(k)u(k|k) + \ldots \\ &+ b_{n_B}(k)u(k-n_B+3) \\ &- a_1(k)\hat{y}(k+2|k) - a_2(k)\hat{y}(k+1) - a_3(k)y(k) - \ldots \\ &- a_{n_A}(k)y(k-n_A+3) + d(k)\end{aligned} \tag{22}$$

$$\vdots$$

Predictions can be expressed in a compact form as functions of future control increments

$$\hat{y}(k+1|k) = s_1(k)\triangle u(k|k) + \ldots \tag{23}$$

$$\hat{y}(k+2|k) = s_2(k)\triangle u(k|k) + s_1(k)\triangle u(k+1|k) + \ldots \tag{24}$$

$$\hat{y}(k+3|k) = s_3(k)\triangle u(k|k) + s_2(k)\triangle u(k+1|k) + s_1(k)\triangle u(k+2|k) + \ldots \tag{25}$$

$$\vdots$$

where step-response coefficients of the linearised model are determined recurrently for $j = 1, \ldots, N$ from

$$s_j(k) = \sum_{i=1}^{\min(j,n_B)} b_i(k) - \sum_{i=1}^{\min(j-1,n_A)} a_i(k)s_{j-i}(k) \tag{26}$$

Using the linearised model (19), it is possible to express the output prediction vector $\hat{\boldsymbol{y}}(k) = [\hat{y}(k+1|k) \ldots \hat{y}(k+N|k)]^T$ as the sum of two parts

$$\hat{\boldsymbol{y}}(k) = \boldsymbol{G}(k)\triangle\boldsymbol{u}(k) + \boldsymbol{y}^0(k) \tag{27}$$

The first part depends only on the future (on future control moves $\triangle\boldsymbol{u}(k)$), the second part is a free trajectory vector $\boldsymbol{y}^0(k) = \left[y^0(k+1|k) \ldots y^0(k+N|k)\right]^T$, which depends only on the past. The dynamic matrix $\boldsymbol{G}(k)$ of dimensionality $N \times N_u$ contains step-response coefficients of the local linear approximation of the nonlinear model

$$\boldsymbol{G}(k) = \begin{bmatrix} s_1(k) & 0 & \ldots & 0 \\ s_2(k) & s_1(k) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ s_N(k) & s_{N-1}(k) & \ldots & s_{N-N_u+1}(k) \end{bmatrix} \tag{28}$$

The dynamic matrix is calculated on-line from the local linearisation of the full nonlinear model taking into account the current state of the process.

Thanks to using the suboptimal prediction (27), the optimisation problem (3) becomes the following quadratic programming task

$$\begin{aligned}
&\min_{\triangle \boldsymbol{u}(k)} \left\{ \left\| \boldsymbol{y}^{\mathrm{ref}}(k) - \boldsymbol{G}(k)\triangle \boldsymbol{u}(k) - \boldsymbol{y}^0(k) \right\|^2 + \left\| \triangle \boldsymbol{u}(k) \right\|^2_{\boldsymbol{\Lambda}} \right\} \\
&\text{subject to} \\
&\quad \boldsymbol{u}^{\min} \leq \boldsymbol{J}\triangle \boldsymbol{u}(k) + \boldsymbol{u}^{k-1}(k) \leq \boldsymbol{u}^{\max} \\
&\quad -\triangle \boldsymbol{u}^{\max} \leq \triangle \boldsymbol{u}(k) \leq \triangle \boldsymbol{u}^{\max} \\
&\quad \boldsymbol{y}^{\min} \leq \boldsymbol{G}(k)\triangle \boldsymbol{u}(k) + \boldsymbol{y}^0(k) \leq \boldsymbol{y}^{\max}
\end{aligned} \tag{29}$$

where $\boldsymbol{y}^{\mathrm{ref}}(k) = \left[y^{\mathrm{ref}}(k+1|k) \ldots y^{\mathrm{ref}}(k+N|k) \right]^T$, $\boldsymbol{y}^{\min} = \left[y^{\min} \ldots y^{\min} \right]^T$, $\boldsymbol{y}^{\max} = \left[y^{\max} \ldots y^{\max} \right]^T$ are vectors of length N, $\boldsymbol{u}^{\min} = \left[u^{\min} \ldots u^{\min} \right]^T$, $\boldsymbol{u}^{\max} = \left[u^{\max} \ldots u^{\max} \right]^T$, $\boldsymbol{u}^{k-1}(k) = \left[u(k-1) \ldots u(k-1) \right]^T$, $\triangle \boldsymbol{u}^{\max} = \left[\triangle u^{\max} \ldots \triangle u^{\max} \right]^T$ are vectors of length N_u, $\boldsymbol{\Lambda} = diag(\lambda_0, \ldots, \lambda_{N_u - 1})$, $\boldsymbol{J}$ is the all ones lower triangular matrix of dimensionality $N_u \times N_u$.

If output constraints are present, the MPC optimisation task (29) may be affected by the infeasibility problem. In such a case the original output constraints have to be softened by using slack variables [29, 47]. Using a quadratic penalty for constraint violations the MPC-NPL optimisation problem is

$$\begin{aligned}
&\min_{\triangle \boldsymbol{u}(k),\ \boldsymbol{\varepsilon}^{\min},\ \boldsymbol{\varepsilon}^{\max}} \{ \left\| \boldsymbol{y}^{\mathrm{ref}}(k) - \boldsymbol{G}(k)\triangle \boldsymbol{u}(k) - \boldsymbol{y}^0(k) \right\|^2 + \left\| \triangle \boldsymbol{u}(k) \right\|^2_{\boldsymbol{\Lambda}} \\
&\qquad + \rho^{\min} \left\| \boldsymbol{\varepsilon}^{\min} \right\|^2 + \rho^{\max} \left\| \boldsymbol{\varepsilon}^{\max} \right\|^2 \} \\
&\text{subject to} \\
&\quad \boldsymbol{u}^{\min} \leq \boldsymbol{J}\triangle \boldsymbol{u}(k) + \boldsymbol{u}^{k-1}(k) \leq \boldsymbol{u}^{\max} \\
&\quad -\triangle \boldsymbol{u}^{\max} \leq \triangle \boldsymbol{u}(k) \leq \triangle \boldsymbol{u}^{\max} \\
&\quad \boldsymbol{y}^{\min} - \boldsymbol{\varepsilon}^{\min} \leq \boldsymbol{G}(k)\triangle \boldsymbol{u}(k) + \boldsymbol{y}^0(k) \leq \boldsymbol{y}^{\max} + \boldsymbol{\varepsilon}^{\max} \\
&\quad \boldsymbol{\varepsilon}^{\min} \geq 0, \quad \boldsymbol{\varepsilon}^{\max} \geq 0
\end{aligned} \tag{30}$$

where slack variables vectors of length N are denoted by $\boldsymbol{\varepsilon}^{\min}$ and $\boldsymbol{\varepsilon}^{\max}$, $\rho^{\min}$ and $\rho^{\max} > 0$ are weights.

All things considered, at each sampling instant k of the MPC-NPL algorithm the structure of which is shown in Fig. 3, the following steps are repeated:

1. Linearisation of the neural model: obtain the matrix $\boldsymbol{G}(k)$.
2. Find the nonlinear free trajectory $\boldsymbol{y}^0(k)$ using the neural model.
3. Solve the quadratic programming task (30) to find the control policy $\triangle \boldsymbol{u}(k)$.
4. Implement the first element of the obtained policy $u(k) = \triangle u(k|k) + u(k-1)$.
5. Set $k := k + 1$, go to step 1.

The formulation of the MPC-NPL algorithm is general, different neural models can be used: MLP structures [20, 21, 22, 26, 28, 47, 48], RBF models [27], neural Hammerstein [25] and Wiener models [23]. Thanks to a simple, regular nature of these models, the implementation of the algorithm is easy. For example, taking into account the structure of the MLP neural model defined by (6) and (7),

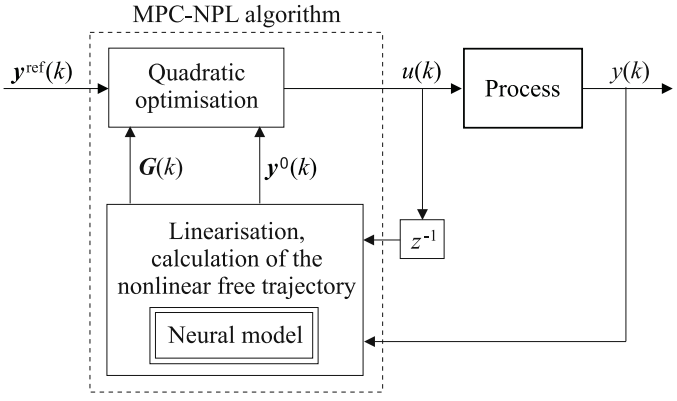

Fig. 3. The structure of the MPC-NPL algorithm

coefficients of the linearised model $a_l(k) = -\frac{\partial f(\bar{\boldsymbol{x}}(k))}{\partial y(k-l)}$ and $b_l(k) = \frac{\partial f(\bar{\boldsymbol{x}}(k))}{\partial u(k-l)}$ are calculated easily from

$$a_l(k) = -\sum_{i=1}^{K} w_i^2 \frac{\mathrm{d}\varphi(z_i(\bar{\boldsymbol{x}}(k)))}{\mathrm{d}z_i(\bar{\boldsymbol{x}}(k))} w_{i,I_u+l}^1 \tag{31}$$

where $l = 1, \ldots, n_A$, and

$$b_l(k) = \begin{cases} 0 & \text{if } l = 1, \ldots, \tau - 1 \\ \sum_{i=1}^{K} w_i^2 \frac{\mathrm{d}\varphi(z_i(\bar{\boldsymbol{x}}(k)))}{\mathrm{d}z_i(\bar{\boldsymbol{x}}(k))} w_{i,l-\tau+1}^1 & \text{if } l = \tau, \ldots, n_B \end{cases} \tag{32}$$

The linearisation point $\bar{\boldsymbol{x}}(k) = [\bar{u}(k-\tau) \ldots \bar{u}(k-n_B)\ \bar{y}(k-1) \ldots \bar{y}(k-n_A)]^T$ is determined by past input and output signals (measurements) corresponding to the arguments of the nonlinear model (5). If hyperbolic tangent is used as the nonlinear transfer function φ in the hidden layer of the neural model, one has $\frac{\mathrm{d}\varphi(z_i(\bar{\boldsymbol{x}}(k)))}{\mathrm{d}z_i(\bar{\boldsymbol{x}}(k))} = 1 - \tanh^2(z_i(\bar{\boldsymbol{x}}(k)))$.

For optimisation of the future control policy a linearised model is used. Although the free trajectory can be also calculated using the linearised model, it is a better idea to use the full nonlinear neural model. The nonlinear free trajectory $y^0(k+p|k)$ over the prediction horizon, i.e. for $p = 1, \ldots, N$, is calculated recursively from

$$y^0(k+p|k) = w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi(z_i^0(k+p|k)) + d(k) \tag{33}$$

Quantities $z_i^0(k+p|k)$ are determined from (7) assuming no changes in the control signal from the sampling instant k onwards. One has

$$z_i^0(k+p|k) = w_{i,0}^1 + \sum_{j=1}^{I_{uf}(p)} w_{i,j}^1 u(k-1) + \sum_{j=I_{uf}(p)+1}^{I_u} w_{i,j}^1 u(k-\tau+1-j+p) + \sum_{j=1}^{I_{yp}(p)} w_{i,I_u+j}^1 y^0(k-j+p|k) + \sum_{j=I_{yp}(p)+1}^{n_A} w_{i,I_u+j}^1 y(k-j+p) \tag{34}$$

where $I_{uf}(p) = \max(\min(p-\tau+1, I_u), 0)$ and $I_{yp}(p) = \min(p-1, n_A)$. Using (6) and (18), the unmeasured disturbance is estimated from

$$d(k) = y(k) - \left(w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi(z_i(k)) \right) \tag{35}$$

6 Training of Neural Models for MPC

The neural model is usually trained off-line. During on-line control in MPC the model is not retrained. On-line model adaptation, although discussed in the literature [2, 50], has a limited applicability and in practice is not popular. For training a set of data is given (i.e. the input sequence $u(1), \ldots, u(S)$ and the output sequence $y(1), \ldots, y(S)$ where S is the number of samples) collected from measurements of the process. The objective of identification is to find the topology and parameters of the neural model in such a way that a predefined performance index which describes the accuracy of the model is minimised. A yet another set (the test set) is used in order to assess generalisation abilities of the model.

Identification consists of three phases: model structure selection, training and assessment. Typically, various models with different topology (K) and different order of dynamics (n_A, n_B, τ) should be trained and evaluated. The order of dynamics can be determined analytically [7]. The model finally chosen for MPC should have a relatively small number of parameters and be precise.

Typically, during model training the following Sum of Squared Errors (SSE) performance function is minimised

$$\text{SSE} = \sum_{k \in \text{data set}} (y(k|k-1) - y(k))^2 \tag{36}$$

where $y(k|k-1)$ denotes the output of the model for the sampling instant k calculated using signals up to the sampling instant $k-1$ whereas $y(k)$ is the real value of the process output variable collected during the identification experiment.

Training is an unconstrained optimisation task in which the SSE performance index is minimised. Gradient-based algorithms are usually used, gradients of

the SSE function are calculated analytically using the backpropagation scheme [11]. The optimisation direction can be found by different nonlinear optimisation methods: the steepest descent, the conjugate gradient methods (Polak-Ribiere, Fletcher-Reeves), the quasi-Newton algorithms (DFP, BFGS) or the Levenberg-Marquardt method [6]. Because of fast convergence and robustness BFGS and Levenberg-Marquardt algorithms are recommended. The optimisation problem is likely to be non-convex, the minimised objective function may have many local minima. Hence, in practice, for a given model structure the gradient-based algorithm is usually initialised randomly and the training procedure is repeated a few times (the multi-start approach). Alternatively, for initialisation global optimisation can be used (e.g. the genetic algorithm, simulated annealing).

Two configurations of dynamic models can be used: the one-step ahead prediction configuration (the series-parallel model) and the simulation configuration (the parallel model) [36]. In the first case the current value of the output signal, $y(k)$, is a function of past input and output values (i.e. real values measured at previous sampling instants). In the second case current and future output values are calculated recurrently, without using real output measurements. Naturally, in MPC the model must be used recurrently since the prediction over the prediction horizon N is considered. For the model (5) one has

$$y(k|k) = f(u(k-\tau), \ldots, u(k-n_B), y(k-1), \ldots, y(k-n_A)) \tag{37}$$

$$\begin{aligned} y(k+1|k) = f(&u(k-\tau+1), \ldots, u(k-n_B+1), \\ &y(k|k), \ldots, y(k-n_A+1|k)) \end{aligned} \tag{38}$$

$$\begin{aligned} y(k+2|k) = f(&u(k-\tau+2), \ldots, u(k-n_B+2), \\ &y(k+1|k), \ldots, y(k-n_A+2|k)) \end{aligned} \tag{39}$$

$$\vdots$$

Bearing in mind the role of the model in MPC for long-range prediction, it is obvious that recurrent training should be used [15,36,37]. In the SSE performance function the output of the model $y(k|k-1)$ should be calculated recurrently. An alternative formulation of the performance function can be also used in which all predictions over the whole horizon N are taken into account for all data samples

$$\mathrm{SSE} = \sum_{k \in \text{data set}} \sum_{p=1}^{N} (y(k+p|k-1) - y(k+p))^2 \tag{40}$$

Example: Modelling and MPC of a Polymerisation Reactor

The process under consideration is a polymerisation reaction taking place in a jacketed continuous stirred tank reactor [9] depicted in Fig. 4. The reaction is the free-radical polymerisation of methyl methacrylate with azo-bis-isobutyronitrile as initiator and toluene as solvent. The output $NAMW$ (Number Average Molecular Weight) [kg kmol^{-1}] is controlled by manipulating the inlet initiator

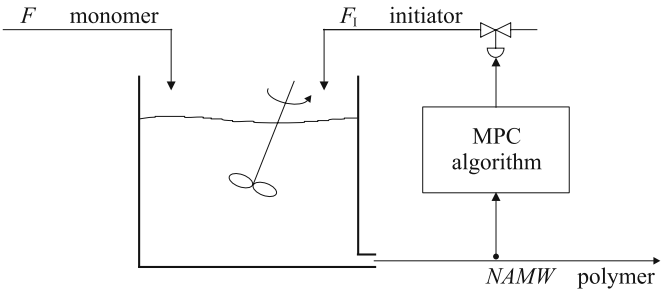

Fig. 4. The polymerisation reactor control system structure

flow rate F_{I} [$\mathrm{m}^3\ \mathrm{h}^{-1}$]. The flow rate F [$\mathrm{m}^3\mathrm{h}^{-1}$] of the monomer is a disturbance. Properties of the considered process are highly nonlinear. The reactor is frequently used as a benchmark [9, 26, 47].

The continuous-time fundamental model of the polymerisation reactor [9] is comprised of four nonlinear ordinary differential equations

$$\frac{\mathrm{d}C_{\mathrm{m}}(t)}{\mathrm{d}t} = -\left[Z_{\mathrm{P}}\exp\left(\frac{-E_{\mathrm{P}}}{RT}\right) + Z_{\mathrm{f_m}}\exp\left(\frac{-E_{\mathrm{f_m}}}{RT}\right)\right]C_{\mathrm{m}}(t)P_0(t) - \frac{F(t)C_{\mathrm{m}}(t)}{V} + \frac{F(t)C_{\mathrm{m_{in}}}}{V} \tag{41}$$

$$\frac{\mathrm{d}C_{\mathrm{I}}(t)}{\mathrm{d}t} = -Z_{\mathrm{I}}\exp\left(\frac{-E_{\mathrm{I}}}{RT}\right)C_{\mathrm{I}}(t) - \frac{F(t)C_{\mathrm{I}}}{V} + \frac{F_{\mathrm{I}}(t)C_{\mathrm{I_{in}}}}{V} \tag{42}$$

$$\frac{\mathrm{d}D_0(t)}{\mathrm{d}t} = \left[0.5Z_{\mathrm{T_c}}\exp\left(\frac{-E_{\mathrm{T_c}}}{RT}\right) + Z_{\mathrm{T_d}}\exp\left(\frac{-E_{\mathrm{T_d}}}{RT}\right)\right]P_0^2(t) + Z_{\mathrm{f_m}}\exp\left(\frac{-E_{\mathrm{f_m}}}{RT}\right)C_{\mathrm{m}}(t)P_0(t) - \frac{F(t)D_0(t)}{V} \tag{43}$$

$$\frac{\mathrm{d}D_{\mathrm{I}}(t)}{\mathrm{d}t} = M_{\mathrm{m}}\left[Z_{\mathrm{P}}\exp\left(\frac{-E_{\mathrm{P}}}{RT}\right) + Z_{\mathrm{f_m}}\exp\left(\frac{-E_{\mathrm{f_m}}}{RT}\right)\right]C_{\mathrm{m}}(t)P_0(t) - \frac{F(t)D_{\mathrm{I}}(t)}{V} \tag{44}$$

where

$$P_0(t) = \sqrt{\frac{2f^*C_{\mathrm{I}}(t)Z_{\mathrm{I}}\exp\left(\frac{-E_{\mathrm{I}}}{RT}\right)}{Z_{\mathrm{T_d}}\exp\left(\frac{-E_{\mathrm{T_d}}}{RT}\right) + Z_{\mathrm{T_c}}\exp\left(\frac{-E_{\mathrm{T_c}}}{RT}\right)}} \tag{45}$$

and one algebraic output equation

$$NAMW(t) = \frac{D_{\mathrm{I}}(t)}{D_0(t)} \tag{46}$$

Parameters of the fundamental model are given in Table 1. The initial operating conditions are: $F_{\mathrm{I}} = 0.028328\ \mathrm{m}^3\ \mathrm{h}^{-1}$, $F = 1\ \mathrm{m}^3\ \mathrm{h}^{-1}$, $NAMW = 20000$ kg

Table 1. Parameters of the fundamental model

Parameter	Value	Parameter	Value
$C_{\mathrm{I}_{\mathrm{in}}}$	8 kmol m^{-3}	R	8.314 kJ kmol^{-1} K^{-1}
$C_{\mathrm{m}_{\mathrm{in}}}$	6 kmol/m^{-3}	T	335 K
$E_{\mathrm{T_c}}$	$2.9442 \cdot 10^3$ kJ kmol^{-1}	$Z_{\mathrm{T_c}}$	$3.8223 \cdot 10^{10}$ m^3 kmol^{-1} h^{-1}
$E_{\mathrm{T_d}}$	$2.9442 \cdot 10^3$ kJ kmol^{-1}	$Z_{\mathrm{T_d}}$	$3.1457 \cdot 10^{11}$ m^3 kmol^{-1} h^{-1}
$E_{\mathrm{f_m}}$	$7.4478 \cdot 10^4$ kJ kmol^{-1}	$Z_{\mathrm{f_m}}$	$1.0067 \cdot 10^{15}$ m^3 kmol^{-1} h^{-1}
E_{I}	$1.2550 \cdot 10^5$ kJ kmol^{-1}	Z_{I}	$3.7920 \cdot 10^{18}$ h^{-1}
E_{P}	$1.8283 \cdot 10^4$ kJ kmol^{-1}	Z_{P}	$1.7700 \cdot 10^9$ m^3 kmol^{-1} h^{-1}
f^*	0.58	V	0.1 m^3
M_{m}	100.12 kg kmol^{-1}		

kmol^{-1}, $C_{\mathrm{m}} = 5.3745$ kmol m^{-3}, $C_{\mathrm{I}} = 2.2433 \cdot 10^{-1}$ kmol m^{-3}, $D_0 = 3.1308 \cdot 10^{-3}$ kmol m^{-3}, $D_{\mathrm{I}} = 6.2616 \cdot 10^{-1}$ kmol m^{-3}.

For identification the fundamental model (41)–(46) is used as the real process, it is simulated open-loop in order to obtain two sets of data, namely training and test data sets depicted in Fig. 5. Both sets contain 2000 samples, the sampling time is 1.8 min. The output signal contains small measurement noise. During calculations the system of differential equations comprising the fundamental model is solved using the Runge-Kutta RK45 method.

The classical MLP neural model of the process is used. A number of model candidates are trained to assess the order of dynamics and the number of hidden nodes. Finally, the second-order model is chosen containing $K = 6$ hidden nodes with the hyperbolic tangent transfer function in the hidden layer

$$y(k) = f(u(k-2), y(k-1), y(k-2)) \tag{47}$$

Because input and output process variables have different orders of magnitude, they are scaled as $u = 100(F_{\mathrm{I}} - F_{\mathrm{I0}})$, $y = 0.0001(NAMW - NAMW_0)$ where $F_{\mathrm{I0}} = 0.028328$, $NAMW_0 = 20000$ correspond to the initial operating point.

During model training the SSE performance function (36) is minimised. Because models used in MPC have to be able to make good predictions not only one

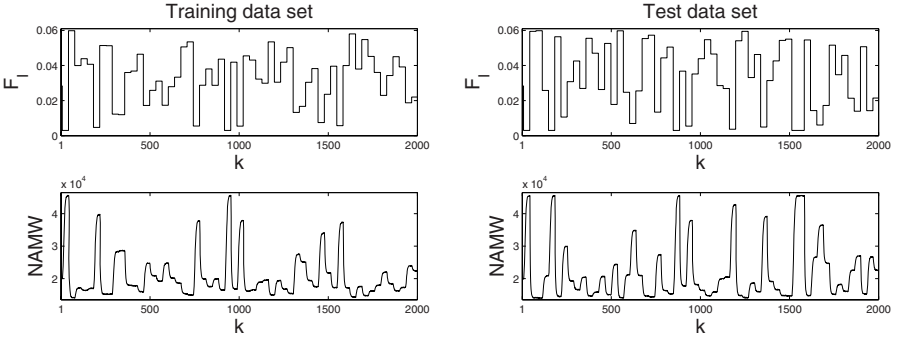

Fig. 5. Training and test data sets

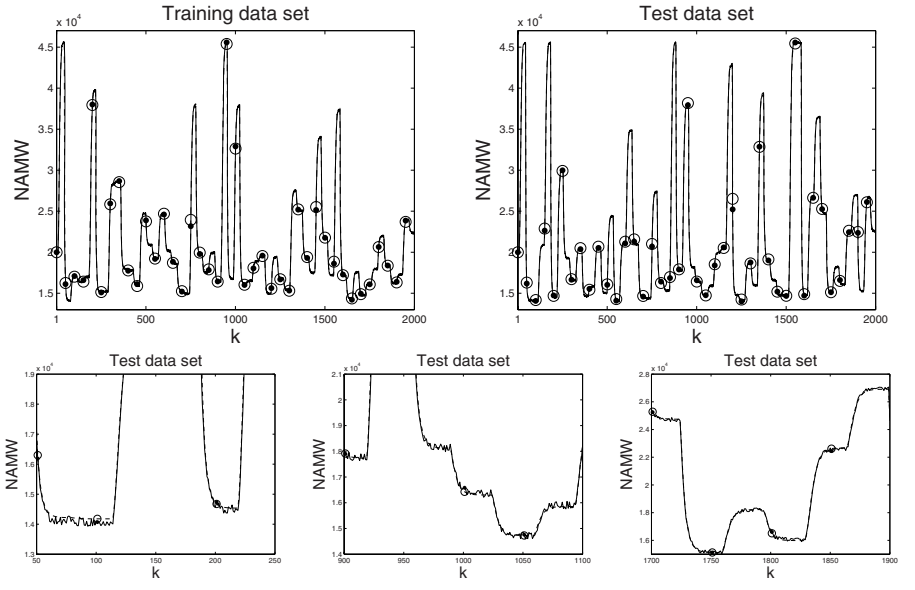

Fig. 6. The process (*solid line with dots*) vs. the neural model (*dashed line with circles*) for training and test data sets: complete data sets (*top*) and enlarged fragments of the test data set (*bottom*)

step ahead, but over the whole prediction horizon, neural networks are trained as recurrent Output Error (OE) models which make it possible to properly calculate multi-step ahead predictions [37]. The SSE performance function is minimised using the BFGS optimisation algorithm [6]. For each neural model topology training is repeated 10 times, weights of neural networks are initialised randomly. Fig. 6 shows the output of the process and the output of the chosen neural model for both training and test data sets. The accuracy of the neural model is very high. For the training data set $\mathrm{SSE} = 5.559159 \cdot 10^{-1}$, for the test data set $\mathrm{SSE} = 1.190907 \cdot 10^{0}$.

To demonstrate very high accuracy of the neural model the linear model

$$y(k) = b_2 u(k-2) - a_1 y(k-1) - a_2 y(k-2) \tag{48}$$

is found. The linear model has exactly the same arguments as the neural one. Fig. 7 shows the output of the process and the output of the linear model for both data sets. Unfortunately, because the process is really nonlinear, the accuracy of the linear model is low. For the training data set $\mathrm{SSE} = 3.402260 \cdot 10^{2}$, for the test data set $\mathrm{SSE} = 6.255822 \cdot 10^{2}$. Comparing properties of both models, one can expect that it is better to use in MPC the neural model rather than the linear one.

Compared MPC strategies are:

a) the linear MPC algorithm based on the linear model (48),

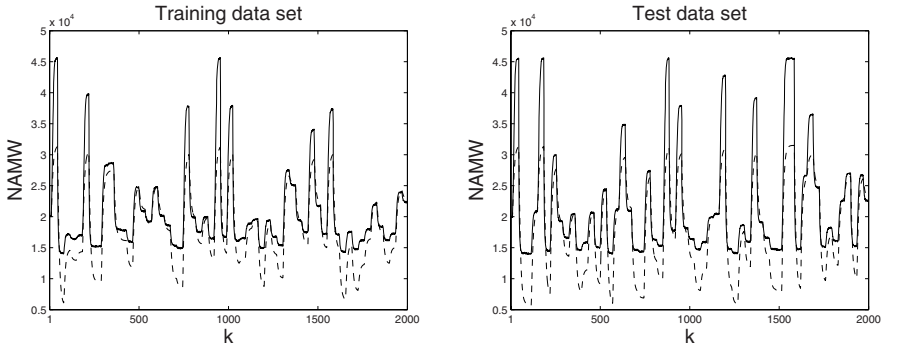

Fig. 7. The process (*solid line*) vs. the linear model (*dashed line*) for training and test data sets

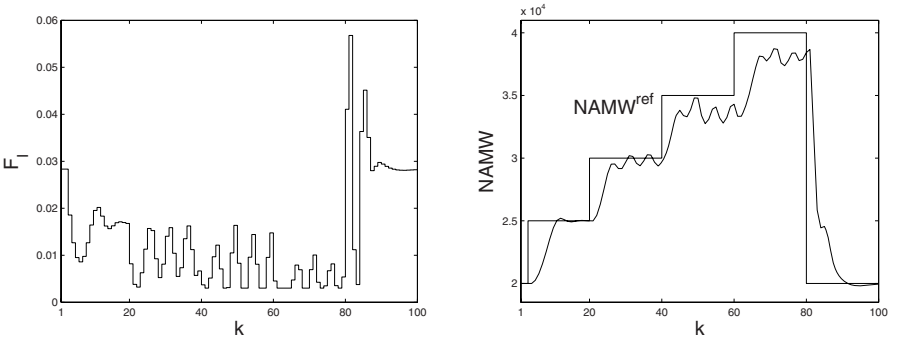

Fig. 8. Simulation results of the MPC algorithm based on the linear model

b) the nonlinear MPC-NPL algorithm based on the neural model (47),
c) the nonlinear MPC-NO algorithm based on the same neural model (47).

All algorithms are implemented in Matlab. The MPC algorithm based on the linear model and the MPC-NPL algorithm use the quadratic programming procedure whereas the MPC-NO algorithm uses the Sequential Quadratic Programming (SQP) nonlinear optimisation routine. As the initial point for MPC-NO nonlinear optimisation $N_u - 1$ control values calculated at the previous sampling instant and not applied to the process are used. The fundamental model (41)–(46) is used as the real process, it is solved using the Runge-Kutta RK45 method.

Horizons of all compared MPC algorithms are $N = 10$, $N_u = 3$, the weighting coefficients $\lambda_p = 0.2$. (As far as choosing parameters of MPC the reader is referred to the literature [29,44,45,47].) The manipulated variable is constrained: $F_{\mathrm{I}}^{\min} = 0.003$, $F_{\mathrm{I}}^{\max} = 0.06$, the sampling time is 1.8 min.

Simulation results of the MPC algorithm based on the linear model are depicted in Fig. 8. As the reference trajectory ($NAMW^{\mathrm{ref}}$) five set-point changes

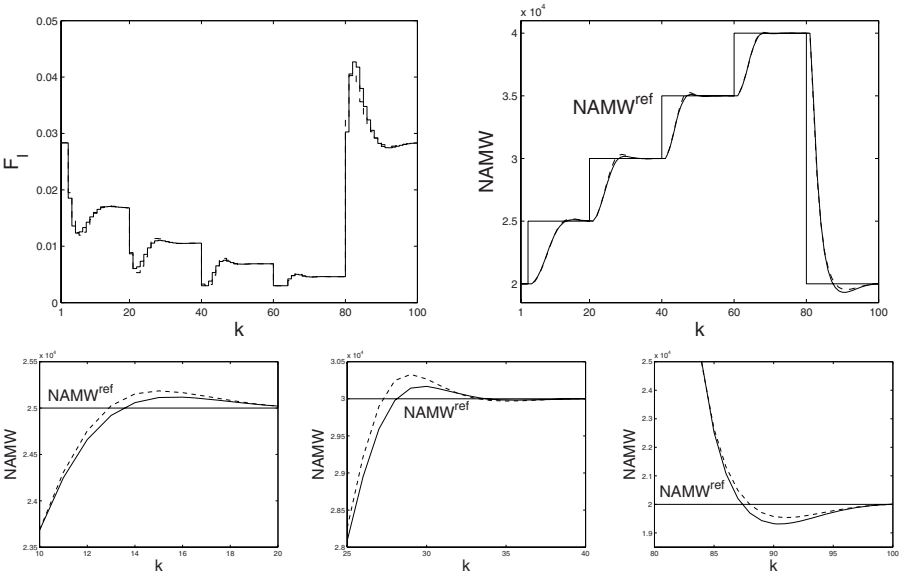

Fig. 9. Simulation results of MPC-NPL (*solid*) and MPC-NO (*dashed*) algorithms based on the same neural model: the whole simulation (*top*) and enlarged fragments (*bottom*)

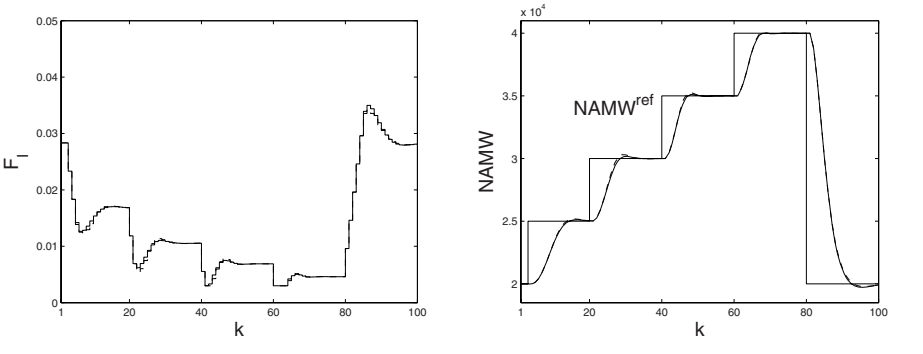

Fig. 10. Simulation results of MPC-NPL (*solid*) and MPC-NO (*dashed*) algorithms based on the same neural model with constraints imposed on increments of the manipulated variable, $\triangle F_{\mathrm{I}}^{\max} = 0.005\ \mathrm{m}^3/\mathrm{h}$

are considered. The linear algorithm works well only for the smallest set-point change, whereas for bigger ones the system becomes unstable. Simulation results of MPC-NPL and MPC-NO algorithms based on the same neural model are depicted in Fig. 9. Both nonlinear algorithms are stable. Moreover, the closed-loop performance obtained in the suboptimal MPC-NPL algorithm with quadratic programming is very close to that obtained in the computationally demanding

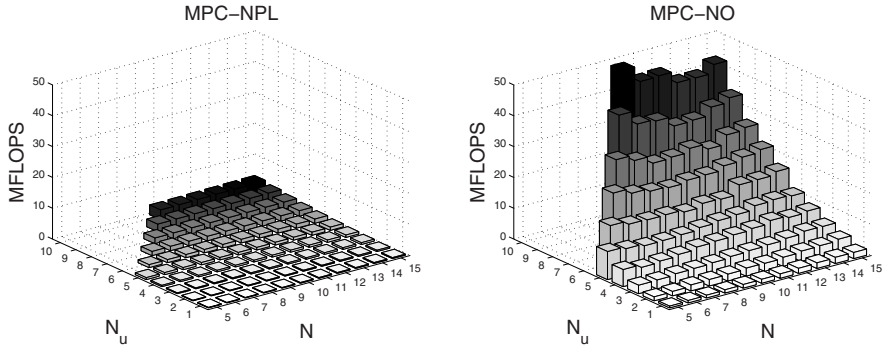

Fig. 11. The computational complexity (MFLOPS) of MPC-NPL (*left*) and MPC-NO (*right*) algorithms based on the same neural model

Table 2. The computational complexity (MFLOPS) of MPC-NPL and MPC-NO algorithms based on the same neural model

Algorithm	N	$N_u = 1$	$N_u = 2$	$N_u = 3$	$N_u = 4$	$N_u = 5$	$N_u = 10$
MPC-NPL	5	0.131	0.176	0.275	0.422	0.637	–
MPC-NO	5	0.560	1.278	2.782	5.560	9.101	–
MPC-NPL	10	0.221	0.285	0.405	0.573	0.808	3.198
MPC-NO	10	1.262	2.640	4.110	6.779	8.896	48.358
MPC-NPL	15	0.320	0.413	0.563	0.764	1.033	3.628
MPC-NO	15	1.968	3.993	5.846	8.547	11.182	42.367

MPC-NO approach, in which a nonlinear optimisation problem has to be solved on-line at each sampling instant.

Simulation results of both nonlinear algorithms with constraints imposed on increments of the manipulated variable $\triangle F_{\mathrm{I}}^{\max} = 0.005$ m^3/h are shown in Fig. 10. Such constraints are very important when changes in the reference trajectory are big, they take into account the actuator's limitations. In comparison with Fig. 9, additional constraints result in a slightly slower output profile, but technological restrictions are rigorously taken into account.

Table 2 shows the influence of control and prediction horizons on the computational complexity of MPC-NPL and MPC-NO algorithms (in terms of floating point operations MFLOPS) for $N = 5, 10, 15$, $N_u = 1, 2, 3, 4, 5, 10$. Fig. 11 depicts the computational complexity for $N = 5, \ldots, 15$, $N_u = 1, \ldots, 10$. In general, the suboptimal MPC-NPL algorithm is considerably less computationally demanding than the MPC-NO strategy. The minimal computational complexity reduction (MPC-NPL vs. MPC-NO) for all considered combinations of horizons is 4.28 times, the maximal is 15.37 times, the average reduction factor is 10.59.

7 Specialised Neural Models for MPC

In MPC the explicit model of the process is directly used to predict its future behaviour and to determine the optimal control policy. As a result, MPC algorithms are very model-based, the control performance is determined by the accuracy of predictions calculated by the model. The role of the model in MPC control cannot be ignored during model structure selection and identification. Naturally, the model has to be able to make good predictions of future behaviour of the process over the whole prediction horizon.

According to the general prediction equation (17)

$$\hat{y}(k+p|k) = y(k+p|k) + d(k) \tag{49}$$

predictions are calculated by means of the model used for the sampling instant $k+p$ at the current sampling instant k. The unmeasured disturbance estimation $d(k)$ is also calculated (18) using the model of the process. Using the prediction equation and the general nonlinear neural model (5) which represents any model discussed so far (MLP, RBF neural Hammerstein or Wiener), output predictions over the prediction horizon ($p = 1, \ldots, N$) are calculated recurrently from

$$\begin{aligned}\hat{y}(k+p|k) = f(&\underbrace{u(k-\tau+p|k), \ldots, u(k|k)}_{I_{uf}(p)}, \underbrace{u(k-1), \ldots, u(k-n_B+p)}_{I_u - I_{uf}(p)}, \\ &\underbrace{\hat{y}(k-1+p|k), \ldots, \hat{y}(k+1|k)}_{I_{yp}(p)}, \underbrace{y(k), \ldots, y(k-n_A+p)}_{n_A - I_{yp}(p)}) + d(k)\end{aligned} \tag{50}$$

Predictions $\hat{y}(k+p|k)$ depend on $I_{uf}(p) = \max(\min(p-\tau+1, I_u), 0)$ future values of the control signal (i.e. decision variables of the MPC algorithm), where $I_u = n_B - \tau + 1$, $I_u - I_{uf}(p)$ values of the control signal applied to the plant at previous sampling instants, $I_{yp}(p) = \min(p-1, n_A)$ future output predictions and $n_A - I_{yp}(p)$ plant output signal values measured at previous sampling instants. For prediction in MPC the classical NARX model (5) has to be used recurrently, because predictions depend on predictions calculated for previous sampling instants within the prediction horizon.

Quite frequently, neural models are trained using the rudimentary backpropagation algorithm which yields one-step ahead predictors. Intuitively, they are not suited to be used recurrently in MPC for long-range prediction (50) since the prediction error is propagated. It is particularly important in the case of noise, model inaccuracies and underparameterisation, i.e. the order of the model is usually significantly lower than the order of the real process or even the proper model order is unknown.

To solve the problem resulting from the inaccuracy of one-step ahead predictors in MPC two general approaches can be recommended. First of all, specialised recurrent training algorithms for neural models can be used (Section 6), but they are significantly more computationally demanding in comparison with one-step ahead predictor training. Moreover, obtained models can be sensitive to noise. An alternative is to choose the structure of the model in such a way that its

role in MPC is not ignored. In this work two specialised neural structures are discussed: multi-models and structured models. In both cases, thanks to the nature of these models, the prediction error is not propagated. Both model types are easily trained as one-step ahead predictors.

7.1 Neural FIR Models

The easiest way of avoiding the necessity of using the model recurrently is to use Nonlinear Finite Impulse Response (NFIR) models

$$y(k) = f(\boldsymbol{x}(k)) = f(u(k-\tau), \ldots, u(k-n_B)) \tag{51}$$

where $f : \Re^{n_B-\tau+1} \longrightarrow \Re$. For the NFIR model, output predictions

$$\begin{aligned}\hat{y}(k+p|k) = & f(\underbrace{u(k-\tau+p|k), \ldots, u(k|k)}_{I_{uf}(p)}, \underbrace{u(k-1), \ldots, u(k-n_B+p)}_{I_u - I_{uf}(p)}) \\ & + d(k)\end{aligned} \tag{52}$$

depend only on past and future control signals. For prediction in MPC the NFIR model is not used recurrently, the prediction error is not propagated.

The NFIR model can be realised by the MLP or RBF neural network. The output of the MLP NFIR model is given by (6). In contrast to the classical MLP neural model (7), sums of inputs of hidden nodes ($i = 1, \ldots, K$) depend only on the input signal

$$z_i(k) = w_{i,0}^1 + \sum_{j=1}^{I_u} w_{i,j}^1 u(k-\tau+1-j) \tag{53}$$

Weights of the network are denoted by $w_{i,j}^1$, $i = 1, \ldots, K$, $j = 0, \ldots, n_B - \tau + 1$, and w_i^2, $i = 0, \ldots, K$, for the first and the second layer, respectively, $I_u = n_B - \tau + 1$.

Unfortunately, because of its nature, the NFIR model usually needs a high order of dynamics (determined by n_B), much higher than the classical model of a similar accuracy which depends on both past input and output signals (5).

7.2 Neural Multi-models

In the multi-model approach [24] one independent neural model is used for each sampling instant within the prediction horizon. For the sampling instant $k+1$ the following model is used

$$y(k+1) = f_1(u(k-\tau+1), \ldots, u(k-n_B), y(k), \ldots, y(k-n_A)) \tag{54}$$

For the sampling instant $k+2$ the model is

$$y(k+2) = f_2(u(k-\tau+2), \ldots, u(k-n_B), y(k), \ldots, y(k-n_A)) \tag{55}$$

In a similar way independent submodels are formulated for all sampling instants within the prediction horizon. For the sampling instant $k+N$ the model is

$$y(k+N) = f_N(u(k-\tau+N), \ldots, u(k-n_B), y(k), \ldots, y(k-n_A)) \tag{56}$$

In general, for $p = 1, \ldots, N$, all submodels can be expressed in a compact form

$$\begin{aligned} y(k+p) = f_p(\boldsymbol{x}(k+p|k)) = \\ f_p(u(k-\tau+p), \ldots, u(k-n_B), y(k), \ldots, y(k-n_A)) \end{aligned} \tag{57}$$

The multi-model is comprised of N neural networks which calculate predictions for consecutive sampling instants within the prediction horizon. Each network realises the function $f_p : \Re^{\min(n_B+p-\tau+1, n_B)+\max(p-\tau+1,0)+n_A+1} \longrightarrow \Re$.

Predictions calculated from the multi-model are

$$\hat{y}(k+p|k) = y(k+p|k) + d(k+p|k) \tag{58}$$

for $p = 1, \ldots, N$. Independent disturbance estimations are

$$d(k+p|k) = y(k) - f_p(k|k-1) \tag{59}$$

where $y(k)$ is measured while $f_p(k|k-1)$ is calculated from the multi-model used for the sampling instant k.

Using (57) and (58), output predictions for $p = 1, \ldots, N$ calculated from the multi-model are

$$\begin{aligned} \hat{y}(k+p|k) = f_p(\underbrace{u(k-\tau+p|k), \ldots, u(k|k)}_{I_{uf}(p)}, \underbrace{u(k-\max(\tau-p,1)), \ldots, u(k-n_B)}_{I_{up}(p)} \\ \underbrace{y(k), \ldots, y(k-n_A))}_{n_A+1} + d(k+p|k) \end{aligned} \tag{60}$$

where $I_{uf}(p) = \max(p-\tau+1, 0)$, $I_{up}(p) = n_B - \max(\tau-p, 1)+1$. Analogously as in the case of the classical NARX model (5), predictions calculated by means of the multi-model (57) depend on $I_{uf}(p)$ future values of the control signal, $I_{up}(p)$ values of the control signal applied to the plant at previous sampling instants and on n_A+1 values of the plant output signal measured at previous sampling instants. Unlike classical predictions (50), they do not depend on predictions calculated for previous sampling instants within the prediction horizon. As a result, the multi-model is not used recurrently, the prediction error is not propagated. Fig. 12 depicts the structure of the multi-model used for prediction in MPC.

Neural multi-model training needs finding independent N submodels. They are trained separately by means of the standard backpropagation algorithm which yields one-step ahead predictors. It is possible because for prediction one independent neural submodel is used for each sampling instant within the prediction horizon and predictions do not depend on previous predictions.

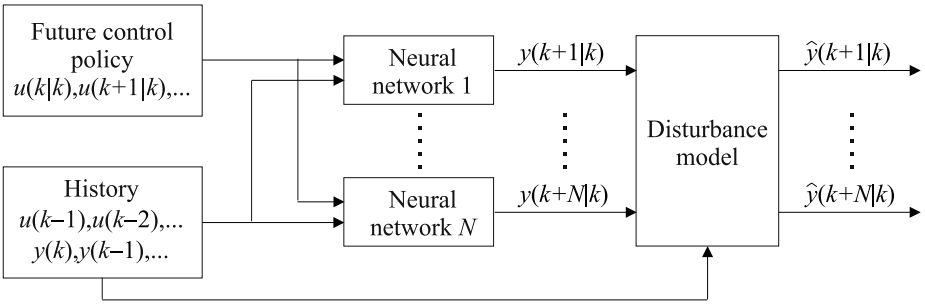

Fig. 12. The structure of the neural multi-model used for prediction in MPC

As submodels both MLP and RBF neural networks can be used. These neural networks realise functions f_p, $p = 1, \ldots, N$ in (57). Outputs of MLP submodels for the sampling instant $k + p$, $p = 1, \ldots, N$ are

$$y(k+p|k) = f_p(\boldsymbol{x}(k+p|k)) = w_0^{2,p} + \sum_{i=1}^{K^p} w_i^{2,p} \varphi(z_i^p(k+p|k)) \tag{61}$$

where $z_i^p(k+p|k)$ are sums of inputs of the i^{th} hidden node, K^p is the number of hidden nodes. Recalling the prediction of the multi-model (60) one has

$$\begin{aligned} z_i^p(k+p|k) = & w_{i,0}^{1,p} + \sum_{j=1}^{I_{uf}(p)} w_{i,j}^{1,p} u(k-\tau+1-j+p|k) \\ & + \sum_{j=1}^{I_{up}(p)} w_{i,I_{uf}(p)+j}^{1,p} u(k-\max(\tau-p,1)+1-j) \\ & + \sum_{j=1}^{n_{A+1}} w_{i,I_{uf}(p)+I_{up}(p)+j}^{1,p} y(k+1-j) \end{aligned} \tag{62}$$

Weights are denoted by $w_{i,j}^{1,p}$, $i = 1, \ldots, K^p$, $j = 0, \ldots, \max(p-\tau+1, 0) - \max(\tau-p, 1) + n_A + n_B + 2$, and $w_i^{2,p}$, $i = 0, \ldots, K^p$, for the first and the second layer, respectively, p indicates the submodel, $p = 1, \ldots, N$.

The MPC-NPL algorithm based on MLP multi-models is detailed in [24].

7.3 Structured Neural Models

Rewriting the model (5) for sampling instants $k-1, \ldots, k-N+1$ one has

$$\begin{aligned} y(k-1) = f(&u(k-\tau-1), \ldots, u(k-n_B-1), \\ &y(k-2), \ldots, y(k-n_A-1)) \end{aligned} \tag{63}$$

$$\vdots$$

$$y(k-N+2) = f(u(k-\tau-N+2),\ldots,u(k-n_B-N+2), \quad (64)$$
$$y(k-N+1),\ldots,y(k-n_A-N+2))$$
$$y(k-N+1) = f(u(k-\tau-N+1),\ldots,u(k-n_B-N+1), \quad (65)$$
$$y(k-N),\ldots,y(k-n_A-N+1))$$

Using (65), the quantity $y(k-N+2)$ given by (64) can be expressed as

$$y(k-N+2) = f(u(k-\tau-N+2),\ldots,u(k-n_B-N+2), \quad (66)$$
$$f(u(k-\tau-N+1),\ldots,u(k-n_B-N+1),$$
$$y(k-N),\ldots,y(k-n_A-N+1)),$$
$$y(k-N),\ldots,y(k-n_A-N+2))$$

which can be rewritten as the function

$$y(k-N+2) = f_{N-2}(u(k-\tau-N+2),\ldots,u(k-n_B-N+1), \quad (67)$$
$$y(k-N),\ldots,y(k-n_A-N+1))$$

Model arguments rearrangement can be repeated for all quantities $y(k-N+2),\ldots,y(k)$, giving functions $f_{N-2},\ldots,f_0$. Finally, one has

$$y(k) = f(u(k-\tau),\ldots,u(k-n_B), \quad (68)$$
$$f_1(u(k-\tau-1),\ldots,u(k-n_B-N+1),\ldots,$$
$$y(k-N),\ldots,y(k-n_A-N+1)),\ldots,$$
$$f_{n_A}(u(k-\tau-n_A),\ldots,u(k-n_B-N+1),$$
$$y(k-N),\ldots,y(k-n_A-N+1)))$$

which can be rewritten as the function

$$y(k) = f_0(u(k-\tau),\ldots,u(k-n_B-N+1), \quad (69)$$
$$y(k-N),\ldots,y(k-n_A-N+1))$$

The equation (69) represents the structured model, $f_0 : \Re^{n_A+n_B-\tau+N} \longrightarrow \Re$.

Using the general prediction equation (17), output predictions calculated from the structured model (69) are

$$\hat{y}(k+p|k) = f_0(\underbrace{u(k-\tau+p|k),\ldots,u(k|k)}_{I_{uf}(p)},\underbrace{u(k-1),\ldots,u(k-n_B-N+1+p)}_{I_u-I_{uf}(p)}, \quad (70)$$
$$\underbrace{y(k-N+p),\ldots,y(k-n_A-N+1+p)}_{n_A}) + d(k)$$

For the structured model $I_u = n_B + N - \tau$. Predictions depend on $I_{uf}(p) = \max(p-\tau+1,0)$ future values of the control signal and $I_u - I_{uf}(p)$ values of the control signal applied to the plant at previous sampling instants. Unlike classical predictions (50), they do not depend on predictions calculated for previous

sampling instants within the prediction horizon, but only on n_A values of the plant output signal measured at previous sampling instants. As a result, the structured model is not used recurrently, the prediction error is not propagated.

Both MLP and RBF neural networks can be used in the structured model to realise the function f_0 in (69). The output of the structured MLP model is

$$y(k) = f(\boldsymbol{x}(k)) = w_0^2 + \sum_{i=1}^{K} w_i^2 \varphi(z_i(k)) \tag{71}$$

where $z_i(k)$ is the sum of inputs of the i^{th} hidden node. From (69)

$$z_i(k) = w_{i,0}^1 + \sum_{j=1}^{I_u} w_{i,j}^1 u(k-\tau+1-j) + \sum_{j=1}^{n_A} w_{i,I_u+j}^1 y(k-j-N+1) \tag{72}$$

Weights are denoted by $w_{i,j}^1$, $i = 1, \ldots, K$, $j = 0, \ldots, n_A + n_B - \tau + N$, and w_i^2, $i = 0, \ldots, K$, for the first and the second layer, respectively.

The MPC-NPL scheme based on structured MLP models is described in [20].

Example: Multi-Modelling of the Polymerisation Reactor

Two models are trained off-line: the classical NARX model (5)

$$y(k) = f(u(k-2), y(k-1)) \tag{73}$$

and the multi-model (57) for $N = 10$

$$\begin{aligned} y(k+1|k) =& f_1(u(k-1), u(k-2), \\ & y(k), y(k-1)) \end{aligned} \tag{74}$$

$$\begin{aligned} y(k+2|k) =& f_2(u(k|k), u(k-1), u(k-2), \\ & y(k), y(k-1)) \end{aligned} \tag{75}$$

$$\begin{aligned} y(k+3|k) =& f_3(u(k+1|k), u(k|k), u(k-1), u(k-2), \\ & y(k), y(k-1)) \end{aligned} \tag{76}$$

$$\begin{aligned} y(k+4|k) =& f_4(u(k+2|k), u(k+1|k), u(k|k), u(k-1), u(k-2), \\ & y(k), y(k-1)) \end{aligned} \tag{77}$$

$$\vdots$$

$$\begin{aligned} y(k+10|k) =& f_{10}(u(k+8|k), \ldots, u(k|k), u(k-1), u(k-2), \\ & y(k), y(k-1)) \end{aligned} \tag{78}$$

Both models used have the same order of dynamics determined by $\tau = n_B = 2$, $n_A = 1$. To show advantages of the multi-model both models are underparameterised, because in fact the fundamental model (41)–(46) consists of four differential equations. In order to precisely capture the nature of the process, the classical

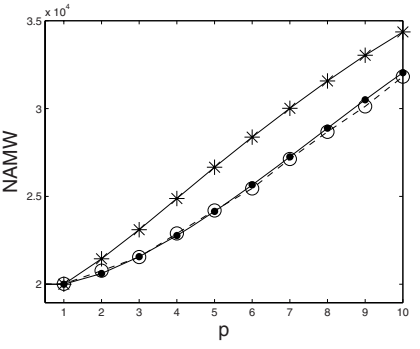

Fig. 13. Step-responses (long-range predictions) calculated recurrently by the classical NARX neural model (*solid line with asterisks*) and by the neural multi-model (*dashed line with circles*) vs. the real process (*solid line with dots*)

neural model should have at least the second order, i.e. $n_A = n_B = \tau = 2$ (in all experiments described so far the model (47), i.e. $y(k) = f(u(k-2), y(k-1), y(k-2))$ is used).

In both model types MLP neural networks are used with the hyperbolic tangent transfer function in hidden layers. All neural models are trained as one-step ahead predictors. The NARX model has $K = 6$ hidden nodes, in the case of the multi-model, six submodels have $K^p = 3$ hidden nodes (for $p = 1, 6, 7, 8, 9, 10$), four submodels have $K^p = 3$ hidden nodes (for $p = 2, 3, 4, 5$). The number of hidden nodes in submodels comprising the multi-model is adjusted in such a way that when trained and tested as one-step ahead predictors they give comparable values of the SSE performance index as the classical NARX model.

To reduce the complexity of models, the Optimal Brain Damage (OBD) pruning algorithm is used [17]. The complexity of the NARX model is reduced by only 16.0% whereas in case of the multi-model in the best case by 34.15%, in the worst case by 17.65%. An average complexity reduction factor is 29.66%.

In light of the application of models in MPC, it is interesting to compare their long range prediction accuracy. Fig. 13 shows step-responses of the process and predictions. The manipulated variable F_I changes at the sampling instant $k = 0$ from 0.028328 to 0.004602, which corresponds to changing the operating point from $NAMW = 20000$ to $NAMW = 40000$. The one-step ahead NARX neural model is used recurrently, it correctly calculates only the prediction for the first sampling instant of the prediction horizon (i.e. for $p = 1$). As a result of underparameterisation, for next sampling instants the prediction error is propagated and consecutive predictions significantly differ from the real process. The neural multi-model is not used recurrently, the prediction error is not propagated. It correctly predicts behaviour of the process over the whole prediction horizon.

To further compare the long-range prediction accuracy and show the potential of using neural multi-models for long-range prediction in MPC, the ratio

$$R_N = \frac{1}{N}\sum_{p=1}^{N}\frac{\sum\limits_{k\in\text{data set}}(y(k+p|k)-y(k+p))^2}{\sum\limits_{k\in\text{data set}}(y_{NARX}(k+p|k)-y(k+p))^2} \tag{79}$$

is considered. It compares the average long-range prediction accuracy of the multi-model (the numerator) and the classical NARX model (the denominator). The output of the classical one-step ahead model used recurrently for long-range prediction is denoted by $y_{NARX}(k+p|k)$, the output of the multi-model is denoted by $y(k+p|k)$, $y(k+p)$ is the real data sample.

If $R_N < 1$ it is a potential for using in MPC multi-models rather than classical models recurrently. The smaller the value of R_N, the worse long-range prediction abilities of the classical model and it is more appropriate to use multi-models. For the training data set $R_{10} = 0.3241$, for the test data set $R_{10} = 0.2298$.

8 The Stabilising MPC-NPL Algorithm

In practice stability of MPC is achieved by adjusting horizons' lengths and penalty factors λ_p. MPC algorithms have gained recognition and have been successfully applied in industry for years before theoretical results concerning their stability analysis appeared. Over the years several MPC algorithms with guaranteed stability have been developed, an excellent review is given in [31]. Usually, stability is guaranteed provided that the nonlinear optimisation is used to solve the MPC optimisation task (3). An additional strong assumption under which stability is guaranteed is the necessity of finding the global solution to the nonlinear MPC optimisation problem at each algorithm iteration. On the other hand, as emphasised in this work, suboptimal MPC algorithms with on-line linearisation and quadratic programming give very good control performance and are computationally efficient. The gap between the practice and the theory is evident. Stability of suboptimal MPC are researched exceptionally infrequently.

It can be easily proved that to guarantee stability it is sufficient to bring the terminal state (i.e. at the end of the prediction horizon) to the origin [31, 32]. In other words, stability is enforced by using in the MPC optimisation task an additional equality terminal constraint

$$x(k+N|k) = 0 \tag{80}$$

The state $x(k) = [u(k-1)\ldots u(k-n_B+1)\ y(k)\ldots y(k-n_A+1)]^T$ [29] corresponds to arguments of the nonlinear model (5). Unfortunately, in nonlinear MPC such a constraint significantly increases the complexity of the nonlinear optimisation problem.

To overcome the necessity of finding the global solution to nonlinear MPC optimisation problem, the dual-mode MPC scheme [34, 46] can be used in which a terminal inequality constraint

$$x(k+N|k) \in \Omega_\alpha \tag{81}$$

guarantees stability. The terminal set Ω_α is a convex neighbourhood of the origin

$$\Omega_\alpha = \left\{ x(k) \in \Re^{n_A+n_B-1} : \|x(k)\|^2_{\boldsymbol{P}} \leq \alpha \right\} \tag{82}$$

where the matrix $\boldsymbol{P} > 0$, $\alpha > 0$. In this approach merely feasibility, rather then optimality, is sufficient to guarantee stability. It means that it is only necessary to find a feasible solution to the nonlinear MPC optimisation problem, this solution does not need to be the global or even a local minimum to guarantee stability of the whole control algorithm. Hence, the dual-mode approach can be efficiently combined with the MPC-NPL algorithm as described in [22]. The formulation of the stabilising MPC-NPL algorithm is general, different neural structures can be used: MLP and RBF models, neural Hammerstein and Wiener models as well as neural multi-models and structured models.

In the dual-model approach the current value of the manipulated variable is calculated in two different ways, the method used depends on the current state of the process. Outside the terminal set Ω_α a nonlinear MPC algorithm is used whereas an additional, usually linear, controller is used inside this set. The objective of the additional feedback controller is to bring the state of the process to the origin. The control law is

$$u(k) = \begin{cases} u(k|k) & \text{if } x(k) \notin \Omega_\alpha \\ \boldsymbol{K}x(k) & \text{if } x(k) \in \Omega_\alpha \end{cases} \tag{83}$$

The nonlinear MPC, which is used if the current state of the plant is outside the terminal set Ω_α, takes into account all the constraints explicitly in the optimisation problem. The linear control law $u(k) = \boldsymbol{K}x(k)$, which is used if the state is inside the terminal set, despite being unconstrained, must never violate constraints. Constraints are used when the terminal set is calculated off-line.

In the stabilising dual-mode approach, the MPC-NPL optimisation problem (30) takes also into account the terminal inequality constraint (81). The main advantage of the algorithm is its suboptimality, i.e. feasibility of the nonlinear optimisation problem (30) is sufficient to guarantee stability. In other words, the determined control sequence $\boldsymbol{u}(k)$ must be always feasible, but it does not need to be the global or even a local minimum of the optimisation problem (30). Determination of the terminal set Ω_α and simulation results of the stabilising dual-model MPC-NPL algorithm are presented in [22].

9 Approximate Neural MPC

Because neural networks are universal approximators, it is an appealing idea to use a network capable of approximating the whole MPC algorithm [4,8,38]. The key idea is to calculate on-line values of the manipulated variable without any optimisation. In other words, the neural network replaces the whole MPC algorithm. The control signal is simply calculated by the neural network as a function of past input and output signals measured at previous sampling instants.

An important advantage of approximate neural MPC is its speed. On the other hand, the main problem is training. At first, a classical MPC algorithm

is developed, for example the MPC-NPL one. Next, a sufficiently large number of simulation should be carried out to cover the whole operation domain (i.e. different initial conditions and reference trajectories). Finally, this data set is used to train a neural model the role of which is to replace the control algorithm.

10 Conclusions

This work emphasises two issues:

1. The proper choice of the model used in MPC is crucial as MPC algorithms are very model-based, the possible control performance is determined by the accuracy of predictions calculated by means of the model.
2. MPC algorithms used in practice should be computationally efficient.

Since fundamental models (complicated systems of nonlinear differential and algebraic equations) are usually not suitable for on-line control and their development is difficult, neural models are recommended. In this work models based on rudimentary MLP and RBF neural networks are discussed, including Hammerstein and Wiener neural structures. All discussed models can be easily used in MPC, without solving any equations which is likely to be computationally demanding and may lead to numerical problems. Neural models are trained using recorded data sets, no technological knowledge is necessary.

The role of the model in MPC cannot be ignored during model structure selection and training. The model has to be able to make good predictions of future behaviour of the process over the whole prediction horizon. Two classes of specialised neural models designed with the specific aim of using them in MPC are discussed: multi-models and structured models. Both these models are trained easily as one-step ahead predictors, but they calculate predictions for the whole prediction horizon without being used recurrently. As a result, the prediction error is not propagated which is particularly important in practice.

In this work the suboptimal MPC-NPL algorithm is recommended. The neural model is successively linearised on-line, the linear approximation is used for prediction and calculation of the future control policy. The MPC-NPL algorithm is computationally efficient, it needs solving on-line a quadratic programming task. It is also demonstrated that the algorithm gives very good control accuracy, comparable to that obtained when nonlinear optimisation is used on-line in MPC.

Acknowledgement. This work was supported by Polish national budget funds 2009-2011 for science as a research project.

References

1. Al-Duwaish, H., Karim, M.N., Chandrasekar, V.: Use of multilayer feedforward neural networks in identification and control of Wiener model. Proceedings IEE, Part D, Control Theory and Applications 143, 225–258 (1996)

2. Alexandridis, A., Sarimveis, H.: Nonlinear adaptive model predictive control based on self-correcting neural network models. AIChE Journal 51, 3495–3506 (2005)
3. Al Seyab, R.K., Cao, Y.: Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation. Journal of Process Control 18, 568–581 (2008)
4. Åkesson, B.M., Toivonen, H.T.: A neural network model predictive controller. Journal of Process Control 16, 937–946 (2006)
5. Arto, V., Hannu, P., Halme, A.: Modeling of chromato-graphic separation process with Wiener-MLP representation. Journal of Process Control 78, 443–458 (2001)
6. Bazaraa, M.S., Sherali, J., Shetty, K.: Nonlinear programming: theory and algorithms. Prentice-Hall, Englewood Cliffs (1999)
7. Bomberger, J.D., Seborg, D.E.: Determination of model order for NARX models directly from input-output data. Journal of Process Control 8, 459–468 (1998)
8. Cavagnari, L., Magni, L., Scattolini, R.: Neural network implementation of nonlinear receding-horizon control. Neural Computing and Applications 8, 86–92 (1999)
9. Doyle, F.J., Ogunnaike, B.A., Pearson, R.K.: Nonlinear model-based control using second-order Volterra models. Automatica 31, 697–714 (1995)
10. El Ghoumari, M.Y., Tantau, H.J.: Non-linear constrained MPC: real-time implementation of greenhouse air temperature control. Computers and Electronics in Agriculture 49, 345–356 (2005)
11. Haykin, S.: Neural networks – a comprehensive foundation. Prentice-Hall, Englewood Cliffs (1999)
12. Henson, M.A.: Nonlinear model predictive control: current status and future directions. Computers and Chemical Engineering 23, 187–202 (1998)
13. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks 2, 359–366 (1989)
14. Hussain, M.A.: Review of the applications of neural networks in chemical process control – simulation and online implmementation. Artificial Intelligence in Engineering 13, 55–68 (1999)
15. Janczak, A.: Identification of nonlinear systems using neural networks and polynomial models. In: A block-oriented approach. LNCIS, vol. 310. Springer, Berlin (2005)
16. Lazar, M., Pastravanu, O.: A neural predictive controller for non-linear systems. Mathematics and Computers in Simulation 60, 315–324 (2002)
17. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. In: Touretzky, D. (ed.) Advances of NIPS2, pp. 598–605. Morgan Kaufmann, San Mateo (1990)
18. Liu, G.P., Kadirkamanathan, V., Billings, S.A.: Predictive control for non-linear systems using neural networks. International Journal of Control 71, 1119–1132 (1998)
19. Luyben, W.L.: Process modelling, simulation and control for chemical engineers. McGraw Hill, New York (1990)
20. Ławryńczuk, M.: Efficient nonlinear predictive control based on structured neural models. International Journal of Applied Mathematics and Computer Science 19 (in press, 2009)
21. Ławryńczuk, M.: Modelling and nonlinear predictive control of a yeast fermentation biochemical reactor using neural networks. Chemical Engineering Journal 145, 290–307 (2008)
22. Ławryńczuk, M., Tadej, W.: A computationally efficient stable dual-mode type nonlinear predictive control algorithm. Control and Cybernetics 145, 99–132 (2008)

23. Ławryńczuk, M.: Suboptimal Nonlinear Predictive Control Based on Neural Wiener Models. In: Dochev, D., Pistore, M., Traverso, P. (eds.) AIMSA 2008. LNCS (LNAI), vol. 5253, pp. 410–414. Springer, Heidelberg (2008)
24. Ławryńczuk, M.: Suboptimal nonlinear predictive control with neural multi-models. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J. (eds.) Computational intelligence: methods and applications, pp. 45–56. Exit, Warsaw (2008)
25. Ławryńczuk, M.: Suboptimal nonlinear predictive control with MIMO neural Hammerstein models. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) IEA/AIE 2008. LNCS (LNAI), vol. 5027, pp. 225–234. Springer, Heidelberg (2008)
26. Ławryńczuk, M.: A family of model predictive control algorithms with artificial neural networks. International Journal of Applied Mathematics and Computer Science 17, 217–232 (2007)
27. Ławryńczuk, M., Tatjewski, P.: A computationally efficient nonlinear predictive control algorithm with RBF neural models and its application. In: Kryszkiewicz, M., Peters, J.F., Rybiński, H., Skowron, A. (eds.) RSEISP 2007. LNCS (LNAI), vol. 4585, pp. 603–612. Springer, Heidelberg (2007)
28. Ławryńczuk, M., Tatjewski, P.: An efficient nonlinear predictive control algorithm with neural models and its application to a high-purity distillation process. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 76–85. Springer, Heidelberg (2006)
29. Maciejowski, J.M.: Predictive control with constraints. Prentice-Hall, Englewood Cliffs (2002)
30. Marlin, T.E.: Process control. McGraw Hill, New York (1995)
31. Mayne, D.Q., Rawlings, J.B., Rao, C.V., Scokaert, P.O.M.: Constrained model predictive control: stability and optimality. Automatica 36, 789–814 (2000)
32. Mayne, D.Q., Michalska, H.: Receding horizon control of nonlinear systems. IEEE Transactions on Automatic Control 35(7), 814–824 (1990)
33. da Cruz Meleiro, L.A., José, F., Zuben, V., Filho, R.M.: Constructive learning neural network applied to identification and control of a fuel-ethanol fermentation process. Engineering Applications of Artificial Intelligence (in press, 2009)
34. Michalska, H., Mayne, D.Q.: Robust receding horizon control of constrained nonlinear systems. IEEE Transactions on Automatic Control 38, 1623–1633 (1993)
35. Morari, M., Lee, J.H.: Model predictive control: past, present and future. Computers and Chemical Engineering 23, 667–682 (1999)
36. Narendra, K.S., Parthasarathy, K.: Identification and control of dynamical systems using neural networks. IEEE Transactions on Neural Networks 1, 4–26 (1990)
37. Nørgaard, M., Ravn, O., Poulsen, N.K., Hansen, L.K.: Neural networks for modelling and control of dynamic systems. Springer, London (2000)
38. Parisini, T., Sanguineti, M., Zoppoli, R.: Nonlinear stabilization by receding-horizon neural regulators. International Journal of Control 70, 341–362 (1998)
39. Pearson, R.K.: Selecting nonlinear model structures for computer control. Journal of Process Control 13, 1–26 (2003)
40. Peng, H., Yang, Z.J., Gui, W., Wu, M., Shioya, H., Nakano, K.: Nonlinear system modeling and robust predictive control based on RBF-ARX model. Engineering Applications of Artificial Intelligence 20, 1–9 (2007)
41. Piche, S., Sayyar-Rodsari, B., Johnson, D., Gerules, M.: Nonlinear model predictive control using neural networks. IEEE Control System Magazine 20, 56–62 (2000)
42. Pottmann, M., Seborg, D.E.: A nonlinear predictive control strategy based on radial basis function models. Computers and Chemical Engineering 21, 965–980 (1997)

43. Qin, S.J., Badgwell, T.A.: A survey of industrial model predictive control technology. Control Engineering Practice 11, 733–764 (2003)
44. Rossiter, J.A.: Model-based predictive control. CRC Press, Boca Raton (2003)
45. Scattolini, R., Bittanti, S.: On the choice of the horizon in long-range predictive control – some simple criteria. Automatica 26, 915–917 (1990)
46. Scokaert, P.O.M., Mayne, D.Q., Rawlings, J.B.: Suboptimal model predictive control (feasibility implies stability). IEEE Transactions on Automatic Control 3, 648–654 (1999)
47. Tatjewski, P.: Advanced control of industrial processes, structures and algorithms. Springer, London (2007)
48. Tatjewski, P., Ławryńczuk, M.: Soft computing in model-based predictive control. International Journal of Applied Mathematics and Computer Science 16, 101–120 (2006)
49. Trajanoski, Z., Wach, P.: Neural predictive control for insulin delivery using the subcu-taneous route. IEEE Transactions on Biomedical Engineering 45, 1122–1134 (1998)
50. Yu, D.L., Yu, D.W., Gomm, J.B.: Neural model adaptation and predictive control of a chemical process rig. IEEE Transactions on Control Systems Technology 14, 828–840 (2006)
51. Yu, D.L., Gomm, J.B.: Implementation of neural network predictive control to a multivariable chemical reactor. Control Engineering Practice 11, 1315–1323 (2003)
52. Zamarreño, J.M., Vega, P., Garcia, L.D., Francisco, M.: State-space neural network for modelling prediction and control. Control Engineering Practice 8, 1063–1075 (2000)