Computer Assisted Mechanics and Engineering Sciences, 9: 55–69, 2002.

# Artificial neural network models for fault detection and isolation of industrial processes

Józef Korbicz and Andrzej Janczak
*Technical University of Zielona Góra, Institute of Control and Computation Engineering*
*ul. Podgórna 50, 65-246 Zielona Góra, Poland*

(Received August 6, 2001)

The paper focuses on using of artificial neural networks in model-based fault detection and isolation. Modelling of a system both at its normal operation conditions and in faulty states is considered and a comparative study of three different methods of system modelling that use a linear model, neural network nonlinear autoregressive with exogenous input model, and neural network Wiener model is presented. Application of these models is illustrated with an example of approximation of a dependence of the juice steam pressure in the stage two on the juice steam pressures in the stages one and three of a five stage sugar evaporator. Parameters of the linear model are estimated with the recursive pseudolinear regression method, whilst the backpropagation and truncated backpropagation through time algorithms are employed for training the neural network models. All the considered models are derived based on the experimental data recorded at the Lublin Sugar Factory.

**Keywords:** fault detection and isolation, neural network models, parametric models, evaporation stations

## 1. Introduction

In the last two decades, model-based Fault Detection and Isolation (FDI) methods have been studied intensively [3, 7, 18]. The idea of the model-based FDI is to compare the output of a system with the output of the corresponding model of the system. These methods require mathematical models of the considered system both at its normal operation, called a nominal model, and faulty conditions, fault models – Fig. 1. Knowing the nominal model, system inputs and outputs are processed to generate residues $e(n)$, defined as a difference between the system output $y(n)$ and the model output $\hat{y}(n)$. A fault is considered to be detected if the residues or their moving averages exceed a given threshold. This is known as decision-making process based on a simple threshold test. Another decision-making techniques employ methods of statistical decision theory such as generalized likelihood ratio or sequential probability ratio testing [3]. Then, to isolate faults, the residuals can be processed by a fault isolation procedure being a decision making or classification process to determine the location and occurrence time of the faults. Another simple approach to fault isolation uses models of the system at its faulty conditions, as it is shown in Fig. 1. In this case, the system output $y(n)$ and outputs of the corresponding models of the system at its faulty conditions $\hat{y}_{f_1}(n), \ldots, \hat{y}_{f_2}(n)$ are compared. If a residual sequence $\{e_{f_j}(n)\}$ generated as a difference between the system output and a model of the system at its faulty conditions, or its moving averages, does not exceed another given threshold, the fault $j$ is considered to be isolated [10]. The main difficulty in fault isolation is that the measuring data are necessary for all expected faults. For complex industrial processes, such as a five-stage evaporation station, the model-based approach to the FDI can be employed not only for the overall system but also for its selected sub-systems. The FDI system, designed based on such an approach, can deliver both a high sensitivity of the fault detection and a high reliability of the fault isolation.

For many complex industrial processes, we cannot formulate a precise theoretical relationship between process input and output. In this case, experimental models can be built from system input

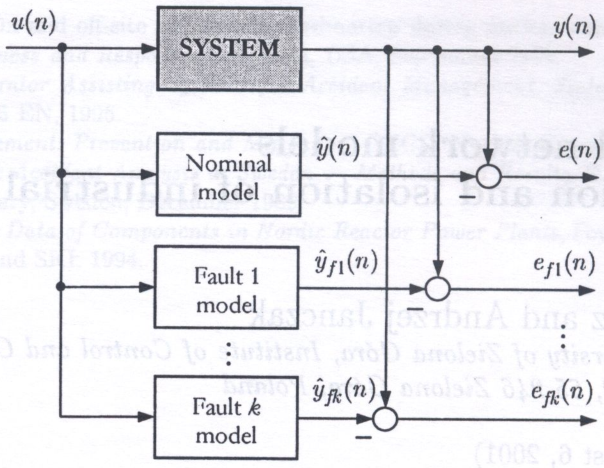

**Fig. 1.** The model-based FDI

and output data. To perform this, large sets of input an output data, usually noise-corrupted, are processed. Such models, whose parameters are simply adjusted to fit to the data and do not reflect any physical parameters are called black-box models.

In the past few years, there has been considerable interest in the application of Artificial Neural Networks (ANN) as models of nonlinear dynamical systems. The most common and successfully proved architecture is the MultiLayer Perceptron architecture (MLP). This is a feedforward network composed of a few feedforward layers of simple processing elements called nodes or neurons. For dynamic systems, to take into account dynamic properties of a system, so called tapped delay lines are used [16, 17]. Provided that the order of the system is known, all the necessary past inputs and outputs are used as inputs to the neural network model. As the MLP is defined by a nonlinear static mapping, it can be trained with the computationally effective BackPropagation (BP) learning algorithm. If one tries to model dynamic systems, including dynamics into a neural network model itself directly seems to be a more natural solution. It can be done by introduction of some feedback connections resulting in neural network structures, called recurrent neural networks. An alternative to the ANN, being nonlinear black-box models, are wavelets and radial basis networks. Incorporation into a model a description that is verbal or imprecise leads to another class of models called the fuzzy models [16].

In the previous works, identification of the five-stage evaporation station as a multidimensional CARIMA model was considered by Lissane Elhaq, Giri, and Unbehauen [13]. A neural network Hammerstein model, neural Nonlinear Auto-Regressive with eXogenous input (NARX) model, and Auto-Regressive with eXogenous input (ARX) model were also used as models of juice steam pressure dynamics [11]. Application of the MLP for modelling the steam juice pressure, juice flow rate, and juice temperature was investigated by Jankowska and Bartyś [12]. Dynamic neural networks as residual generators for different process variables were also considered by Patan and Korbicz [19]. In a three-stage pilot scale evaporator, a neural network approach was also used by Russel et al. [20] for modelling of the stage temperature, concentrate level, and concentrate flow rate.

The motivation of this work is to study different ANN models, capable to model process transient behavior, for fault detection and isolation. The purpose of the work is to develop and compare three different models of the dependence of the juice steam pressure in stage two on the juice steam pressure in stage one and three of a five stage evaporation station. The Output Error (OE), NARX, and neural network Wiener models have been developed based on a real process data recorded at the Lublin Sugar Factory in November 1998.

The remainder of the paper is organized as follows. The theoretical models of the steam juice dynamics are presented in Section 2. Section 3 introduces the ARX, NARX, and Wiener models. The applied identification algorithms, i.e. the recursive least squares, pseudolinear regression, backpropagation, sensitivity method, and backpropagation through time are discussed and derived in

Section 4. Experimental models of the steam pressure dynamics are described in Section 5. Section 6 presents identification results obtained for a real process data. Finally, conclusions are presented in Section 7.

## 2. THEORETICAL MODELS OF JUICE STEAM DYNAMICS

Theoretical approach to the process modelling relies upon deriving a mathematical model of the process by applying the laws of physics. Theoretical models obtained in this way are often too complicated to be used, for example, in control applications. Despite this, the theoretical models are a source of valuable information on the nature of the process. This information can be very useful for identification of experimental mathematical models based on process input-output data. Moreover, the theoretical models can serve as benchmarks for evaluation and verification of the experimental models.

The main task of a multiple-effect evaporator is thickening of the thin sugar juice from the sugar density of approximately 14 to 65–70 Brix units (Bx). The other important tasks are steam generation, waste steam condensation, and supplying waste-heat boilers with water condensate. In a multiple-effect evaporation process, the sugar juice and the saturated steam are fed to the successive stages of the evaporator at gradually decreasing pressure. A flow of steam and sugar juice through the successive stages of the evaporator results in a close relationship between temperatures and pressures of the juice steam in these stages. Many complex physical phenomena and chemical reactions that occur during the thickening of the sugar juice including sucrose decomposition, precipitation of calcium compounds, decomposition of acid amides, etc. make formulation of a theoretical model a difficult task. Moreover, the juice steam temperature and the juice steam pressure depend also on other physical quantities such as the juice rate of flow or the juice temperature.

Modelling of the multiple-effect sugar evaporator is also complicated by the fact that it consists of a number of evaporators connected both in series and in parallel. The theoretical models of a sugar evaporator are derived based on mass and energy balances for all the stages. To perform this, the following assumptions are to be made [13]:

1. Juice and steam are in saturated equilibrium.
2. Both the mass of the steam in the juice chamber and the mass of the steam in the steam chamber are constant.
3. The operation of juice level controllers makes it possible to neglect variations of juice levels.
4. The heat losses to the surrounding environment are negligible.
5. Mixing is perfect.

The model of the dependence of the steam pressure in the the steam chamber of the stage $k$ on the the steam pressure in the the steam chamber of the stage $k-1$, given by Carlos and Corripio [2], has the form

$$P_k = P_{k-1} - \frac{\gamma_k (O_{k-1})^2}{\rho_k^2(T_{k-1}\,,\,P_{k-1})}\,, \qquad k = 2, \ldots, 5, \tag{1}$$

where $P_k$ – the juice steam pressure in the steam chamber of the stage $k$, $\gamma_k$ – the conversion factor, $O_k$ – the steam flow rate from the stage $k$, $\rho_k$ – the steam density at the stage $k$, $T_k$ – the steam temperature at the stage $k$. For the first stage,

$$P_1 = P_0 - \frac{\gamma_1 S^2}{\rho_1^2(T_0\,,\,P_0)}\,, \tag{2}$$

where $S$ – the flow rate of the steam entering the stage $k$.

The model (1) and (2) describes the steady-state behaviour of the process. To include dynamic properties of the process, a modified model described with the differential equations of the first order was proposed,

$$P_k = T_{sk}\frac{\mathrm{d}P_{k-1}}{\mathrm{d}t} + P_{k-1} - \frac{\gamma_k(O_{k-1})^2}{\rho_k^2(T_{k-1}, P_{k-1})}, \tag{3}$$

$$P_1 = T_{s1}\frac{\mathrm{d}P_0}{\mathrm{d}t} + P_0 - \frac{\gamma_1 S^2}{\rho_1^2(T_0 \quad P_0)}, \tag{4}$$

where $T_{sk}$ is the time constant.

Equations (1)–(4) are a part of the overall model of the sucrose juice concentration process. This overall model comprises also the juice flow rate dynamics, juice steam flow rate dynamics, and mass and energy balances for all the stages [13].

## 3. Parametric and neural network models of industrial processes

Parametric models can describe the process behaviour with a finite set of parameters. Examples of the parametric models are differential or difference equations. Nonparametric models, such as an impulse response model, require infinite number of parameters to characterize the process exactly.

The ANN models can be classified as parametric but they usually contain a large number of adjustable parameters called weights. The neuron, called also node or unit, is a simple processing element, which sums the weighted neuron inputs and passes the results through a singular valued activation function [4, 17]. The ANN is a system of such simple processing elements connected into a network by the synaptic weights. There are many different ways, called architectures, in which neurons can be arranged into a network. The ANN architectures can be divided roughly into the following two groups – the feedforward networks, and recurrent (feedback, dynamic) networks. In the feedforward networks, there are no feedback connections, and neurons are usually arranged in a form of layers. The most common feedforward network is the multilayer perceptron. Despite of the fact that the MLP networks does not exhibit any dynamics, they can be used for approximation of nonlinear dynamic systems. This can be achieved if the inputs and outputs of the system, at the past discrete time instances, are chosen as the inputs to the network.

The feedforward architecture can be augmented with feedback connections. The networks of this type are referred as recurrent networks. Many different architectures of the recurrent neural networks has been proposed. Some of the recurrent neural networks have architectures similar to that of the MLP. The examples are memory neural network, the Elman neural network, and the Recurrent MultiLayer Perceptron (RLMP) [16].

### 3.1. ARX and output error models

Complex industrial processes are often approximated with linear models. The ARX model is the most widely applied dynamic stochastic model [14]. The most common approach is to try an ARX model first and if it fails to perform satisfactorily, to examine more complex models. The ARX model has the form

$$y(n) = \frac{B(q^{-1})}{A(q^{-1})}u(n) + \frac{1}{A(q^{-1})}\varepsilon(n) \tag{5}$$

where

$$A(q^{-1}) = 1 + a_1q^{-1} + \cdots + a_{na}q^{-na}, \tag{6}$$

$$B(q^{-1}) = b_1q^{-1} + \cdots + b_{nb}q^{-nb}. \tag{7}$$

The successive values of the noise $\varepsilon(n)$, $n = 1, 2, \ldots$, are realizations of a sequence of independent random variables, $u(n)$ is the system input, $y(n)$ is the system output, and $a_1, \ldots, a_{na}, b_1, \ldots, b_{nb}$ are unknown parameters of the model.

The OE model is characterized by the noise that influences the output directly

$$y(n) = \frac{B(q^{-1})}{A(q^{-1})} u(n) + \varepsilon(n). \tag{8}$$

## 3.2. NARX model

The neural network model of the NARX type has architecture of the MLP (Fig. 2). The MLP is a universal approximator. This means that the MLP with at least one hidden layer can approximate any smooth function to an arbitrary degree of accuracy as the number of hidden layer neurons increases [5]. The MPLs with one hidden layer are most common. In basis formulation, the NARX model with one hidden layer containing $M$ neurons in the hidden layer can be written as

$$\hat{y}(n) = \sum_{i=1}^{M} w_i \Phi[x_i(n)] + w_0 , \tag{9}$$

$$x_i(n) = \left[ \sum_{j=1}^{na+nb} w_{ij} z_j(n) + w_{i0} \right] , \tag{10}$$

$$z_j(n) = u(n - l - j), \qquad j = 1, \ldots, nb, \tag{11}$$

$$z_j(n) = y(n - nb - j), \qquad j = nb + 1, \ldots, na + nb, \tag{12}$$

where $\Phi(\cdot)$ is the activation function, $w_0$ $w_i$ and $w_{i0}$ $w_{ij}$ are the output weights and the hidden layer weights, respectively. Equations (9)–(12) define the serial-parallel NARX model. Replacing the past system outputs $y(n - nb - j)$ with the past model outputs $\hat{y}(n - nb - j)$, a parallel NARX model can be obtained.

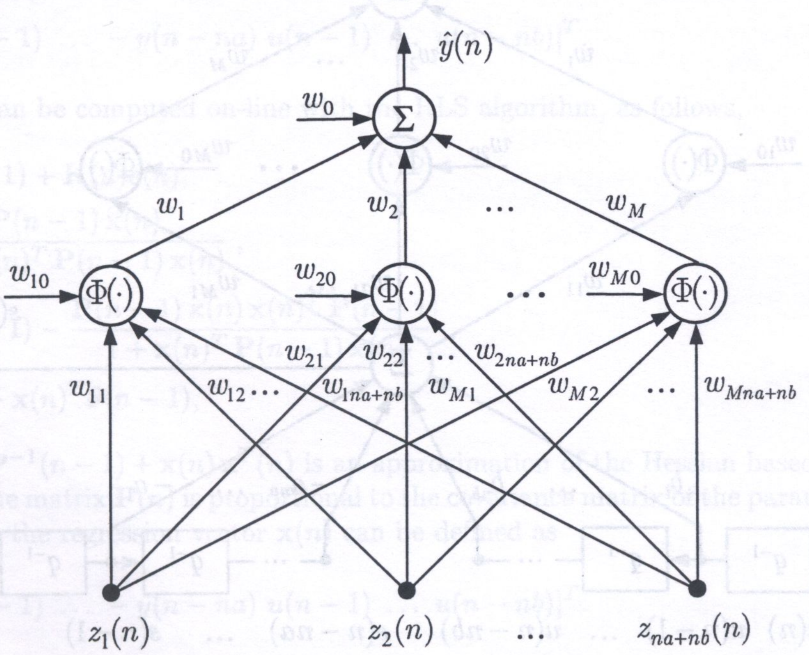

**Fig. 2.** The neural network NARX model

### 3.3. Neural network Wiener model

The Wiener system is composed of a linear dynamic system and a nonlinear static element in cascade. A serial-parallel Wiener model includes not only a model of the linear dynamic system and a model of the static nonlinear element but also an inverse model of the nonlinear element [8, 9]. More simple is architecture of a parallel Wiener model, which does not contain any inverse model. The parallel neural network Wiener model (Fig. 3) consists of a linear neuron with two tapped delay lines modelling the linear dynamical system and a nonlinear perceptron used as a model of the nonlinear static element [1, 9]. The parallel Wiener model is of a recurrent type as it contains a single feedback connection. The output $\hat{y}(n)$ of the parallel Wiener model is is given by

$$\hat{y}(n) = f[s(n), \mathbf{w}] \tag{13}$$

where the output of the linear dynamical model is given by the following difference equation

$$s(n) = -\sum_{m=1}^{na} a_m s(n-m) + \sum_{m=1}^{nb} b_m u(n-m). \tag{14}$$

The static nonlinear function $f(\cdot)$ can be modelled by a MLP with one hidden layer

$$f[s(n), \mathbf{w}] = \sum_{i=1}^{M} w_i \Phi[x_i(n)] + w_0\,, \tag{15}$$

$$x_i(n) = w_{i1} s(n) + w_{i0}\,, \tag{16}$$

$$\mathbf{w} = [w_0 \;\dots\; w_M \; w_{10} \;\dots\; w_{M0} \; w_{11} \;\dots\; w_{M1}]^{\mathrm{T}}, \tag{17}$$

where $w_0$, $w_i$, and $w_{i0}$, $w_{i1}$ are the output weights and the hidden layer weights, respectively.

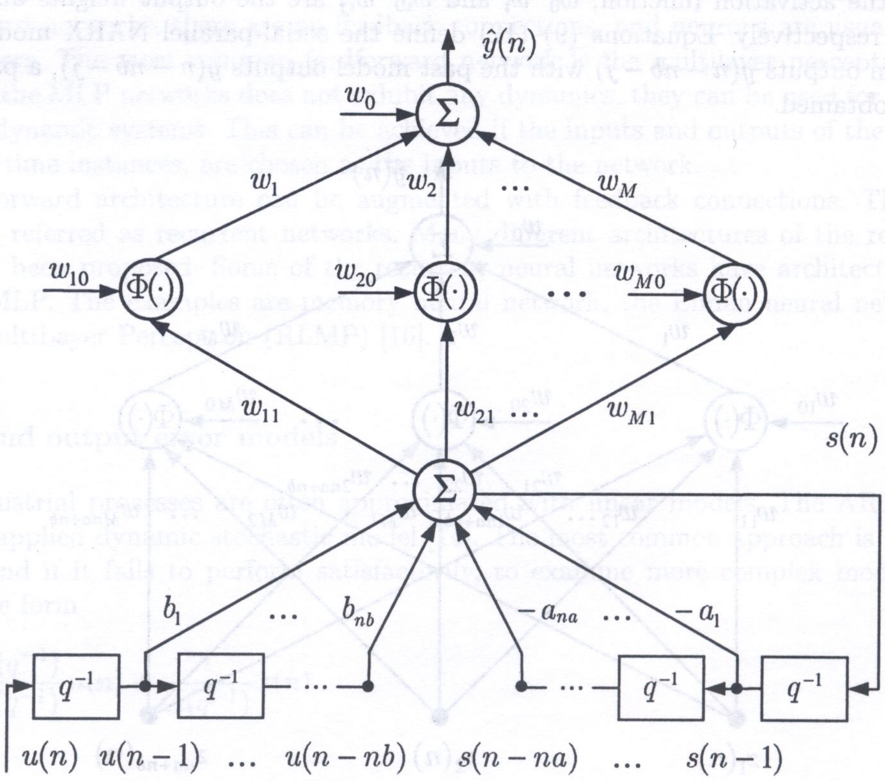

**Fig. 3.** The parallel neural network Wiener model

## 4. Identification algorithms

The considered identification problem can be formulated as follows. Given a set of the system input and output data $\{u(n), y(n)\}$, $n = 1, 2, \ldots, N$, the goal is to train the process model to minimize the error function

$$J_N = \frac{1}{2}\sum_{n=1}^{N}[y(n) - \hat{y}(n)]^2 \tag{18}$$

where $\hat{y}(n)$ is the model output. This can be done by an off-line iterative procedure that uses the whole training set and is called batch learning. For control or filtering applications, it may be more convenient to use another iterative procedure called pattern learning that uses the training data sequentially to minimize

$$J_n = \frac{1}{2}[y(n) - \hat{y}(n)]^2 \tag{19}$$

### 4.1. Recursive least squares and recursive pseudolinear regression

The idea of the Recursive Least Squares (RLS) algorithm is based on computing the new parameter estimate $\boldsymbol{\theta}(n)$ each time new data comes in. This is done by adding a correction vector to the previous estimate $\boldsymbol{\theta}(n-1)$. The RLS algorithm can be applied to the models, such as the ARX model, that are linear in-parameters.

Let the parameter vector $\boldsymbol{\theta}(n)$ has the form

$$\boldsymbol{\theta}(n) = [a_1 \; \ldots \; a_{na} \; b_1 \; \ldots \; b_{nb}]^T. \tag{20}$$

Consequently, the regression vector $\mathbf{x}(n)$ can be written as

$$\mathbf{x}(n) = [-y(n-1) \; \ldots \; -y(n-na) \; u(n-1) \; \ldots \; u(n-nb)]^T. \tag{21}$$

The vector $\boldsymbol{\theta}(n)$ can be computed on-line with the RLS algorithm, as follows,

$$\boldsymbol{\theta}(n) = \boldsymbol{\theta}(n-1) + \mathbf{K}(n)e(n), \tag{22}$$

$$\mathbf{K}(n) = \frac{\mathbf{P}(n-1)\,\mathbf{x}(n)}{1+\mathbf{x}(n)^T\mathbf{P}(n-1)\,\mathbf{x}(n)}, \tag{23}$$

$$\mathbf{P}(n) = \mathbf{P}(n-1) - \frac{\mathbf{P}(n-1)\,\mathbf{x}(n)\,\mathbf{x}(n)^T\,\mathbf{P}(n-1)}{1+\mathbf{x}(n)^T\,\mathbf{P}(n-1)\,\mathbf{x}(n)}, \tag{24}$$

$$e(n) = y(n) - \mathbf{x}(n)^T\boldsymbol{\theta}(n-1), \tag{25}$$

where $\mathbf{P}^{-1}(n) = \mathbf{P}^{-1}(n-1) + \mathbf{x}(n)\,\mathbf{x}^T(n)$ is an approximation of the Hessian based an the already processed data. The matrix $\mathbf{P}(n)$ is proportional to the covariance matrix of the parameter estimates. For the OE model the regression vector $\mathbf{x}(n)$ can be defined as

$$\mathbf{x}(n) = [-\hat{y}(n-1) \; \ldots \; -\hat{y}(n-na) \; u(n-1) \; \ldots \; u(n-nb)]^T \tag{26}$$

As the regression vector $\mathbf{x}(n)$ depends now on the previous parameter vectors $\boldsymbol{\theta}(n-1), \ldots, \boldsymbol{\theta}(n-na)$, the OE model is not linear in-parameters. Application of (22)–(25) to the OE model results in a parameter estimation method known as the Recursive PseudoLinear Regression (RPLR) [14].

## 4.2. Training ANN models with gradient-based learning algorithms

Gradient-based optimization methods are widely used for weight update of neural network models. For serial-parallel models of a static nature, the gradient of their output variables can be computed with the computationally effective BP algorithm. Computation of the gradient for parallel models, which are dynamical systems themselves and their actual output depends not only on model weights but also on their past outputs, is a much more complicated task. Two common methods are the BackPropagation Through Time (BPTT) and the Sensitivity Method (SM), called also the dynamic backpropagation or real time recurrent learning [15]. The BPTT is an extension of the BP algorithm. The basic idea of the BPTT is to unfold the recurrent model back in time. The unfolded network is no more of a recurrent type and can be trained in a way similar to the BP algorithm. The gradient calculation with the BPTT is exact but the unfolding procedure has to be repeated for all time instances $n = 1, 2, \ldots, N$ where $N$ is the number of training data samples. For the BPTT in its basis formulation, both the computational complexity and the memory requirements are high as $N$ is usually quite large. Therefore, an approximate version of the BPTT – truncated BPTT, in which unfolding in time is restricted to only the past $K$ steps, is more feasible in practice.

The SM avoids unfolding of the model back in time. The partial derivatives of the model output with respect to the weights are calculated solving a set of linear difference equation on-line by simulation. The number of these linear difference equations, called also sensitivity models, is equal to the number of weights. In comparison with the BPTT, the SM requires much less memory and its computational complexity does not depend on $N$. However, the SM requires more computation effort than the BP. In both off-line and on-line learning procedures, the weights are updated along the negative gradient of the error function. For pattern learning, updating rule for the weight $w_m$ is

$$w_m(n) = w_m(n-1) - \eta \frac{\partial J_n}{\partial w_m(n-1)} \tag{27}$$

where $n$ is the iteration number and $\eta$ is the learning rate.

The Wiener model contains a linear part which is a model of the linear dynamical system. Thus, the well-known Least Squares (LS) or the RLS methods can be employed to update the parameters $a_1, \ldots, a_{na}, b_1, \ldots,$ and $b_{nb}$ [1, 6, 9]. In this case, the overall learning algorithm combines the BP, SM, or BPTT, for a nonlinear part of the model, with the LS or RLS for the linear part of the model.

### *4.2.1. The NARX model. The backpropagation method*

To compute the gradient of the error function (19), partial derivatives of the model output with respect to all the weights are necessary. Differentiation of (9) gives

$$\frac{\partial \hat{y}(n)}{\partial w_i} = \Phi[x_i(n)], \tag{28}$$

$$\frac{\partial \hat{y}(n)}{\partial w_0} = 1, \tag{29}$$

and taking into account (10)

$$\frac{\partial \hat{y}(n)}{\partial w_{ij}} = \frac{\partial \hat{y}(n)}{\partial \Phi[x_i(n)]} \frac{\partial \Phi[x_i(n)]}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial w_{ij}} = w_i\, \Phi'[x_i(n)]\, z_j(n), \tag{30}$$

$$\frac{\partial \hat{y}(n)}{\partial w_{i0}} = \frac{\partial \hat{y}(n)}{\partial \Phi[x_i(n)]} \frac{\partial \Phi[x_i(n)]}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial w_{i0}} = w_i\, \Phi'[x_i(n)]. \tag{31}$$

### *4.2.2. The parallel Wiener model. The backpropagation method*

The parallel Wiener model does not contain any inverse model of the nonlinear element (Fig. 3). This simplifies the backpropagation learning considerably. On the other hand, as only an approximate gradient is computed, a very slow converge rate can be observed. In the BP method, the dependence of the past linear model outputs $s(n-m)$, $m = 1, 2, \ldots, na$, on the parameters $a_k$ and $b_k$ , is neglected. Hence from (14) it follows that

$$\frac{\partial \hat{y}(n)}{\partial a_k} = \frac{\partial \hat{y}(n)}{\partial s(n)} \frac{\partial s(n)}{\partial a_k} = -s(n-k)\frac{\partial \hat{y}(n)}{\partial s(n)}, \tag{32}$$

$$\frac{\partial \hat{y}(n)}{\partial b_k} = \frac{\partial \hat{y}(n)}{\partial s(n)} \frac{\partial s(n)}{\partial b_k} = u(n-k)\frac{\partial \hat{y}(n)}{\partial s(n)}. \tag{33}$$

The partial derivatives of the parallel Wiener model output with respect to the output of the linear dynamical model are

$$\frac{\partial \hat{y}(n)}{\partial s(n)} = \frac{\partial f[s(n),\mathbf{w}]}{\partial s(n)} = \frac{\partial f[s(n),\mathbf{w}]}{\partial \Phi[x_i(n)]} \frac{\partial \Phi[x_i(n)]}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial s(n)} = \sum_{i=1}^{M} w_i w_{i1}\, \Phi'[x_i(n)]. \tag{34}$$

The partial derivatives of the parallel Wiener model output with respect to the weights $w_{i1}$ , $w_{i0}$ , $w_i$ and $w_0$ are calculated using

$$\frac{\partial \hat{y}(n)}{\partial w_{i1}} = \frac{\partial f[s(n),\mathbf{w}]}{\partial w_{i1}} = \frac{\partial f[s(n),\mathbf{w}]}{\partial \Phi[x_i(n)]} \frac{\partial \Phi[x_i(n)]}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial w_{i1}} = w_i\, \Phi'[x_i(n)]\, s(n), \tag{35}$$

$$\frac{\partial \hat{y}(n)}{\partial w_{i0}} = \frac{\partial f[s(n),\mathbf{w}]}{\partial w_{i0}} = \frac{\partial f[s(n),\mathbf{w}]}{\partial \Phi[x_i(n)]} \frac{\partial \Phi[x_i(n)]}{\partial x_i(n)} \frac{\partial x_i(n)}{\partial w_{i0}} = w_i\, \Phi'[x_i(n)], \tag{36}$$

$$\frac{\partial \hat{y}(n)}{\partial w_i} = \frac{\partial f[s(n),\mathbf{w}]}{\partial w_i} = \Phi[x_i(n)], \tag{37}$$

$$\frac{\partial \hat{y}(n)}{\partial w_0} = \frac{\partial f[s(n),\mathbf{w}]}{\partial w_0} = 1. \tag{38}$$

### *4.2.3. The parallel Wiener model. The sensitivity method*

The SM differs from the BP method markedly in calculation of partial derivatives of the output of linear dynamical model with respect to its parameters. In general, as the SM uses a more accurate evaluation of the gradient than the BP method, a higher convergence rate can be expected.

Assuming that the parameters $a_k$ and $b_k$ do not change and differentiating (14) gives

$$\frac{\partial s(n)}{\partial a_k} = -\sum_{m=1}^{na} a_m \frac{\partial s(n-m)}{\partial a_k} - s(n-k), \tag{39}$$

$$\frac{\partial s(n)}{\partial b_k} = -\sum_{m=1}^{nb} a_m \frac{\partial s(n-m)}{\partial b_k} + u(n-k). \tag{40}$$

The partial derivatives (39) and (40) can be computed on-line by simulation, usually with zero initial conditions. The other partial derivatives are calculated in the SM in the same way as in the BP method. In comparison with the BP method, the SM is only a little more computationally intensive. The increase in computation comes from simulation of $na + nb$ sensitivity models (39) and (40), whereas dynamical models of a low order are identified commonly.

#### *4.2.4. The parallel model. The backpropagation through time method*

In the BPTT, the partial derivatives of the model output with respect to the weights of the nonlinear element model are calculated in the same way as in the BP or SM method from (35)–(38). Similarly, the partial derivatives of the model output with respect to the output of the linear dynamical model are calculated in the same way from (34). The only difference is in computation of the partial derivatives of the linear dynamical model output with respect to its parameters $a_k$ and $b_k$. To perform this, the linear dynamical model is unfolded back in time. The unfolded Wiener model is no more of a recurrent type and can be trained with a method similar to the BP. The detailed description of the truncated BPTT algorithm is given in the Appendix.

## 5. Experimental models of the steam pressure dynamics

From Eq. (3), it follows that while $P_{k-1}$ is the model output, $P_i$, $O_{k-1}$, and $T_{k-1}$ are the model inputs. To solve this nonlinear differential equation numerically, not only the initial conditions, the parameters $\gamma_k$, $T_{sk}$, and the function $\rho_k$ should be known but also all the inputs should be available for measurement. If some of this information is not available one can try to build an experimental model based on the available input and output measurements.

In experimental models of the steam pressure dynamics, considered here, the influence of the steam temperature and the steam flow rate on the the steam pressure is not taken into account. This simplification has two different reasons. The first, the steam flow rate is not measured in the real system. The other, the temperature is not taken into consideration as thermal processes have a slower dynamics in comparison with pressure processes. From a serial structure of the evaporation station, it follows that the steam pressure at stage $i$ depends both on the steam pressure at the preceding stage $i-1$ and succeeding one $i+1$. Therefore, the steam pressures $P_{i-1}$ and $P_{i+1}$ can be assumed to be model inputs.

In the experimental models, the steam pressures in the steam chambers of stages one and three are chosen as the inputs and the steam pressure in stage two as the output.

### 5.1. The linear model

The linear model is composed of the second order OE model and the second order Finite Impulse Response (FIR) model. The FIR model is employed to compensate the influence of the juice steam pressure $P_1(n)$ on the model output $\hat{P}_2(n)$,

$$\hat{P}_2(n) = \frac{B(z^{-1})}{A(z^{-1})} P_3(n) + C(z^{-1})\, P_1(n), \tag{41}$$

where

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2}, \tag{42}$$

$$B(z^{-1}) = b_1 z^{-1} + b_2 z^{-2}, \tag{43}$$

$$C(z^{-1}) = c_1 z^{-1} + c_2 z^{-2}. \tag{44}$$

### 5.2. Neural network NARX model

The neural network NARX model has six inputs: $P_3(n-1)$, $P_3(n-2)$, $\hat{P}_2(n-1)$, $\hat{P}_2(n-2)$, $P_1(n-1)$, $P_1(n-2)$. The one layer MLP network contains 10 nodes of the hyperbolic tangent activation. This results in a model, which contain 81 adjustable connections.

### 5.3. The neural network Wiener model

The neural network model has a similar structure as the linear one (Fig. 4). The only difference is that the second order discrete linear model in the path of the pressure $P_3(n)$ is replaced with a second order Wiener model

$$\hat{P}_2(n) = f\left[\frac{B(z^{-1})}{A(z^{-1})}P_3(n)\right] + C(z^{-1})P_1(n). \tag{45}$$

The static nonlinear function is modelled by the MLP with one hidden layer containing 24 neurons of the hyperbolic tangent activation function. The model contains 79 adjustable parameters.

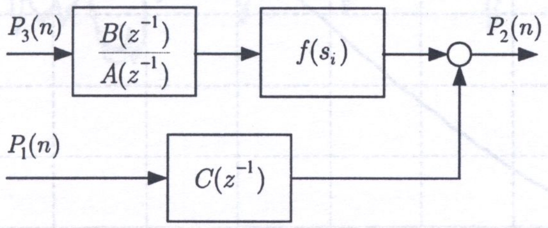

**Fig. 4.** The neural network Wiener model with the FIR compensation of the $P_1(n)$

## 6. Identification results

The data set for model identification and testing was recorded on November 1998 at the Lublin Sugar Factory. The data set contains 18000 measurements collected at the sampling time of 10 s. While the first 10000 measurements has been employed for model estimation, model testing has been performed with the remaining 8000 measurements. As scaling makes the training algorithm numerically robust and leads to faster convergence [17], all the signals have been scaled to zero mean and variance one.

Identification of the models has been performed with recursive algorithms: the RLS for the linear model, the backpropagation for the NARX model, and the truncated BPPT with two unfolding steps for the Wiener model. The optimal value of delay $l = 1$ has been evaluated based on estimation results for the linear model. The following pulse transfer function and the polynomial $C(q^{-1})$ of the linear model have been obtained

$$\frac{B(q^{-1})}{A(q^{-1})} = \frac{0.38846q^{-1} + 0.38793q^{-2}}{1 - 0.12582q^{-1} - 0.12566q^{-2}}, \tag{46}$$

$$C(q^{-1}) = 0.14367q^{-1} + 0.14296q^{-2}. \tag{47}$$

In a recursive learning procedures for both the NARX and Wiener models, the training sequence of 10000 measurements has been employed five times at the learning rate $\eta = 0.01$. The neural network NARX model has the architecture $\mathcal{N}_1(6,10,1)$. The neural network Wiener model contains a model of nonlinear element of the architecture $\mathcal{N}_2(1,24,1)$. Both neural network models contain nodes of hyperbolic tangent activation, in their hidden layers. The obtained nonlinear function $f(\cdot)$ and the step response of the linear dynamic system is shown in Figs. 5 and 6. The pulse transfer function and the polynomial $C(q^{-1})$ of the Wiener model are

$$\frac{B(q^{-1})}{A(q^{-1})} = \frac{0.27596q^{-1} + 0.27436q^{-2}}{1 - 0.10339q^{-1} - 0.10271q^{-2}}, \tag{48}$$

$$C(q^{-1}) = 0.1418q^{-1} + 0.13706q^{-2}. \tag{49}$$

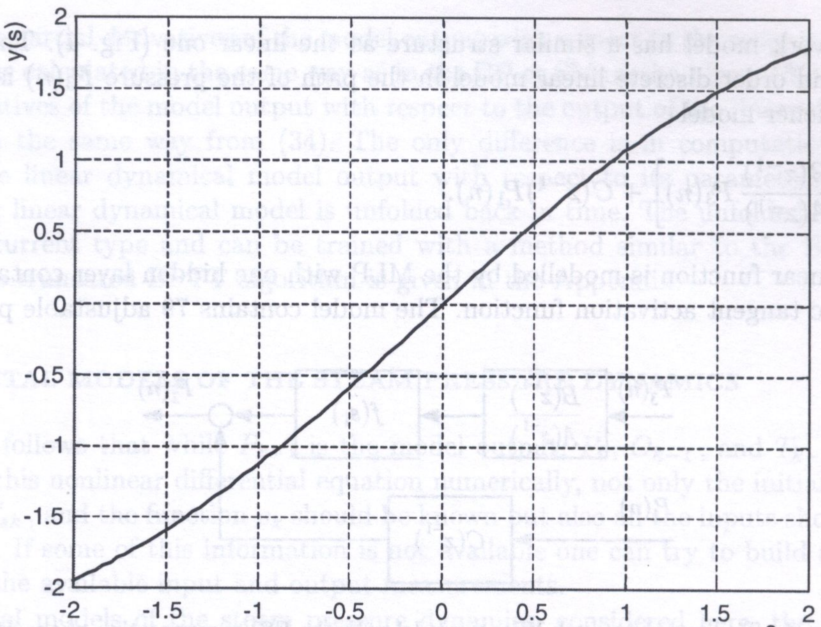

**Fig. 5.** The static nonlinear function of the neural network Wiener model

**Fig. 6.** The step response of the linear part of the neural network Wiener model

**Table 1.** The mean square prediction error of the pressure $P_2$ [kPa]

| | ARX model | NARX model | Wiener model |
|---|---|---|---|
| Training set | 2.103 | 2.591 | 1.804 |
| Testing set | 2.322 | 3.064 | 2.084 |

The juice steam pressure $P_2(n)$ (solid) and neural network Wiener output $\hat{P}_2(n)$ for both the training set and testing set are shown in Figs. 7 and 8. Comparative results of model testing of the mean square prediction error of the pressure $P_2$ are given in Table 1. Compared with the linear model, the NARX model reveals slightly worse, whilst the Wiener model slightly better, prediction properties.

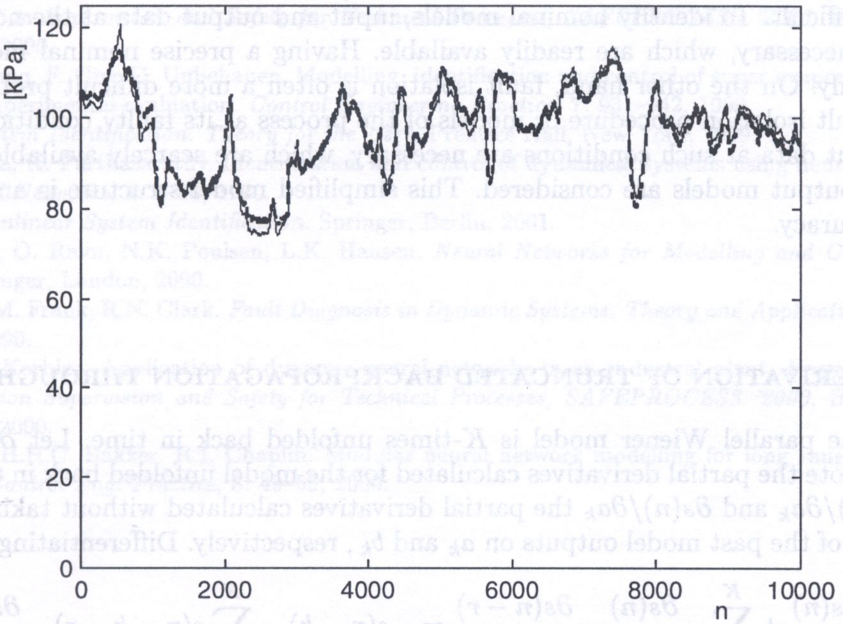

**Fig. 7.** The juice steam pressure $P_2(n)$ (solid) and neural network Wiener output $\hat{P}_2(n)$ – testing with training set

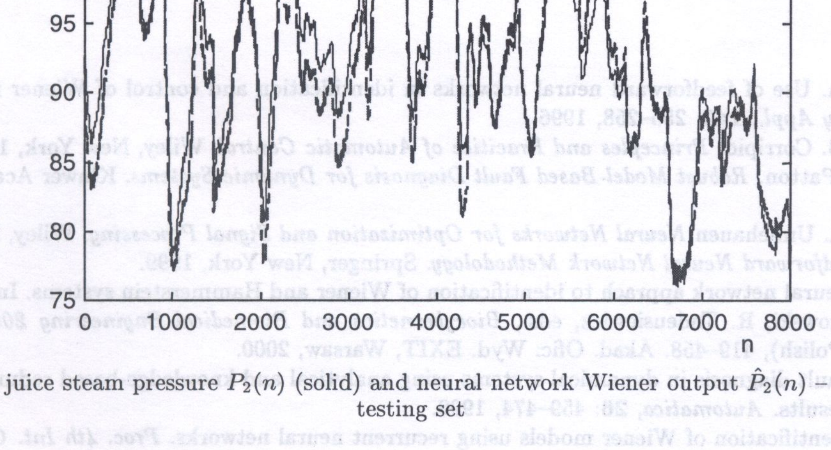

**Fig. 8.** The juice steam pressure $P_2(n)$ (solid) and neural network Wiener output $\hat{P}_2(n)$ – testing with testing set

## 7. Conclusions

Models of industrial processes sub-modules are very useful both for fault detection and fault isolation. The models of the juice steam pressure, considered here, can be characterized with low time constants. Moreover, the steam juice pressure is not controlled automatically. This makes the models very useful for a quick and reliable fault detection. On the other hand, process disturbances, correlated inputs, and unknown order of the system make a practical implementation the model-based FDI methods difficult. To identify nominal models, input and output data at the normal operation conditions are necessary, which are readily available. Having a precise nominal model, faults can be detected easily. On the other hand, fault isolation is often a more difficult problem. To isolate faults with a fault isolation procedure or models of the process at its faulty conditions, the process input and output data at such conditions are necessary, which are scarcely available. In the paper, two input one output models are considered. This simplified model structure is another source of modelling inaccuracy.

## Appendix:

### Gradient derivation of truncated backpropagation through time

Assume that the parallel Wiener model is $K$-times unfolded back in time. Let $\partial^+ s(n)/\partial a_k$ and $\partial^+ s(n)/\partial b_k$ denote the partial derivatives calculated for the model unfolded back in time for $K$ time steps, and $\partial s(n)/\partial a_k$ and $\partial s(n)/\partial a_k$ the partial derivatives calculated without taking into account the dependence of the past model outputs on $a_k$ and $b_k$, respectively. Differentiating (14) we obtain

$$\frac{\partial^+ s(n)}{\partial a_k} = \frac{\partial s(n)}{\partial a_k} + \sum_{r=1}^{K} \frac{\partial s(n)}{\partial s(n-r)} \frac{\partial s(n-r)}{\partial a_k} = -s(n-k) - \sum_{r=1}^{K} s(n-k-r) \frac{\partial s(n)}{\partial s(n-r)}, \tag{50}$$

$$\frac{\partial^+ s(n)}{\partial b_k} = \frac{\partial s(n)}{\partial b_k} + \sum_{r=1}^{K} \frac{\partial s(n)}{\partial s(n-r)} \frac{\partial s(n-r)}{\partial b_k} = u(n-k) + \sum_{r=1}^{K} u(n-k-r) \frac{\partial s(n)}{\partial s(n-r)}. \tag{51}$$

The partial derivatives of the actual output of the linear dynamical model $s(n)$ with respect the delayed outputs $s(n-1), \ldots, s(n-K)$ are

$$\frac{\partial s(n)}{\partial s(n-r)} = -\sum_{m=1}^{na} a_m \frac{\partial s(n-m)}{\partial s(n-r)} = -\sum_{m=1}^{na} a_m \frac{\partial s(n)}{\partial s(n-r+m)} \tag{52}$$

where $\partial s(n-m)/\partial s(n-r) = 0$ for $m > r$.

## References

[1] H. Al-Duwaish. Use of feedforward neural networks in identification and control of Wiener model. *IEE Proc.-Control Theory Appl.*, **143**: 255–258, 1996.

[2] A. Carlos, A.B. Corripio. *Princeples and Pracitice of Automatic Control.* Wiley, New York, 1985.

[3] J. Chen, R.J. Patton. *Robust Model-Based Fault Diagnosis for Dynamic Systems.* Kluwer Academic Publishers, Boston, 1999.

[4] A. Cichocki, R. Unbehauen *Neural Networks for Optimization and Signal Processing.* Wiley, New York, 1993.

[5] T.L. Fine. *Feedforward Neural Network Methodology.* Springer, New York, 1999.

[6] A. Janczak. Neural network apprach to identification of Wiener and Hammerstein systems. In: W. Duch, J. Korbicz, L. Rutkowski, R. Tadeusiewicz, eds., *Biocybernetics and Biomedical Engineering 2000*, Vol. 6. *Neural Networks* (in Polish), 419–458. Akad. Ofic. Wyd. EXIT, Warsaw, 2000.

[7] P.M. Frank. Fault diagnosis in dynamical systems using analytical and knowledge-based redundancy – A survey of some new results. *Automatica*, **26**: 459–474, 1990.

[8] A. Janczak. Identification of Wiener models using recurrent neural networks. *Proc. 4th Int. Conf. Methods and Models in Automation and Robotics, MMAR'97, Międzyzdroje, Poland*, pp. 727–732, 1997.

[9] A. Janczak. Recurrent neural network models for identification of Wiener systems. *Proc. CESA'98 IMACS Multiconference, Symp. Modeling, Analysis and Control, Nabeul-Hammamet, Tunisia*, pp. 965–970, 1998.

[10] A. Janczak, J. Korbicz. Neural network models of Hammerstein systems and their application to fault detection and isolation. *Proc. 14th World Congress of IFAC, Beijing, P.R.C.*, **P**: 91–96, 1999.

[11] A. Janczak. Parametric and neural network models for fault detection and isolation of industrial process submodules. *Prepr. 4th IFAC Symp. Fault Deteection Supervision and Safety for Technical Processes, SAFEPROCESS '2000, Budapest, Hungary*, pp. 348–351, 2000.

[12] A. Jankowska, M. Barty. Application of perceptron neural networks for fault detection. *Prepr. 4th IFAC Symp. Fault Deteection Supervision and Safety for Technical Processes, SAFEPROCESS '2000, Budapest, Hungary*, pp. 203–208, 2000.

[13] S. Lissane Elhaq, F. Giri, H. Unbehauen. Modelling, identification and control of sugar evaporation – theoretical design and experimental evaluation. *Control Engineering Practice*, **7**: 931–942, 1999.

[14] L. Ljung. *System Identification. Theory for the User.* Prentice Hall, New York, 1999.

[15] K.S. Narendra, K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks*, **1**: 4–26, 1990.

[16] O. Nelles. *Nonlinear System Identification.* Springer, Berlin, 2001.

[17] M. Nørgaard, O. Ravn, N.K. Poulsen, L.K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems.* Springer, London, 2000.

[18] R.J. Patton, M. Frank, R.N. Clark. *Fault Diagnosis in Dynamic Systems. Theory and Applications.* Prentice-Hall, New York, 1990.

[19] K. Patan, J. Korbicz. Application of dynamic neural networks in an industral plant. *Prepr. 4th IFAC Symp. Fault Deteection Supervision and Safety for Technical Processes, SAFEPROCESS '2000, Budapest, Hungary*, pp. 186–191, 2000.

[20] N.T. Russel, H.H.C. Bakker, R.I. Chaplin. Modular neural network modelling for long range prediction of an evaporator. *Control Eng. Practice*, **8**: 49–59, 2000.