Nonlinear system identification using neural networks

Pijnenburg, J.L.C.M.

Award date:
1992

Eindhoven University of Technology
Department of Electrical Engineering
Measurement and Control Section

NONLINEAR SYSTEM IDENTIFICATION USING NEURAL NETWORKS

by J.L.C.M. Pijnenburg

M.Sc. Thesis
carried out from September 1991 to April 1992
commissioned by prof.dr.ir. A.C.P.M. Backx
under supervision of dr.ir. A.A.H. Damen and ir. H. Telkamp
date: 14 april 1992

Pijnenburg, J.L.C.M. Nonlinear system identification using neural networks

M.Sc. Thesis, Measurement and Control Section ER, Electrical Engineering, Eindhoven University of Technology, The Netherlands, April 1992

ABSTRACT

Linear system models, which can be obtained from existing linear identification techniques, hold in a fixed operating point. When the process, however, changes its operating point a wide operating range will be passed through and a linear model satisfies no more, a nonlinear model must be used. This clearly occurs in start up, shut down and change over situations.

Multilayer neural networks can 'learn' static nonlinear functions by a given training set of input-output pairs which define this function. When dynamics are included, by means of the feedback of past outputs and the use of past inputs, it is still possible to model nonlinear processes with neural network models. A universal nonlinear system description will be introduced, which includes both a well known linear ARMA model and a nonlinear neural network model. So it is possible to use a priori knowledge about the linear part of the system (possibly from earlier linear identifications) directly as an initialisation. In that case the neural network model only needs to model the remaining nonlinear part of the system.

Simulations show that this approach is able to identify nonlinear systems by estimating the linear and nonlinear parameters simultaneously. A special identification procedure (first the equation error model identification and then the output error model identification) is necessary to obtain a good simulation model in particular when the system output is disturbed by noise. The identification of the water vessel process has proved in conclusion that it is possible to find one nonlinear model for the entire working range in stead of numerous linear models each for a specific working point.

CONTENTS

1 INTRODUCTION

A system working in an operating point, behaves fairly linear and can be modelled by linear models. Much attention has been paid to linear system identification techniques. When a system, however, changes its operating point a wide range has to be passed through and the system can not be described by linear models any more. In such situations it is desirable to have a nonlinear model which describes the system behaviour. This clearly occurs in start up, shut down and change over situations.

In recent years there has been an increasing interest in using neural networks for nonlinear system identification [3,4,6,7,8,9,11,12,17], while neural networks are capable to represent nonlinear relationships. At the section Measurement and Control of the Eindhoven University of technology a study has started to investigate if neural networks can be used for nonlinear system identification. The study includes:

- A literature study, to find out what has been published about neural networks with respect to system identification. Important articles turned out to be [12] from Narendra and Parthasarathy, who give some detailed simulation results, and [20] where the learning algorithm used is discussed.
- The implementation of a learning algorithm (backpropagation). While there is no learning algorithm available, an algorithm has to be written (in C, to speedup the algorithm) and linked to MATLAB.
- The developed algorithm is tested on examples from [12], to verify the algorithm.
- The identification of a water-vessel process using the nonlinear system identification algorithm.

In the next chapter some fundamentals about neural networks are discussed, extra attention is paid to those neural network structures which will be used for nonlinear system modelling. In chapter 3 a universal model is introduced which includes both a linear and a nonlinear model. On the basis of this model a nonlinear system identification using neural networks is discussed. Some simulation results are reviewed in chapter 4 after which the identification of a water-vessel process will be discussed in chapter 5. In the final chapter the results are discussed and recommendations for further research are given.

2 NEURAL NETWORKS

In this chapter some fundamentals about neural networks are discussed. First the structure of a neural networks in general is explained. A so called multilayer neural network is introduced, which is capable to approximate static nonlinear relationships. To learn this input-output relation a learning algorithm is involved. For dynamic system identification applications dynamics must be included into the multilayer neural network, so a dynamic neural network will be explained in the last section.

2.1 Fundamentals of neural networks

A neural network consists of a number of processing elements, also known as neurons, which are connected with each other. The structure of a neural network is defined by the way the processing elements are connected.

The most elementary part of a neural network is the processing element. The structure of a processing element is presented in figure 2.1.

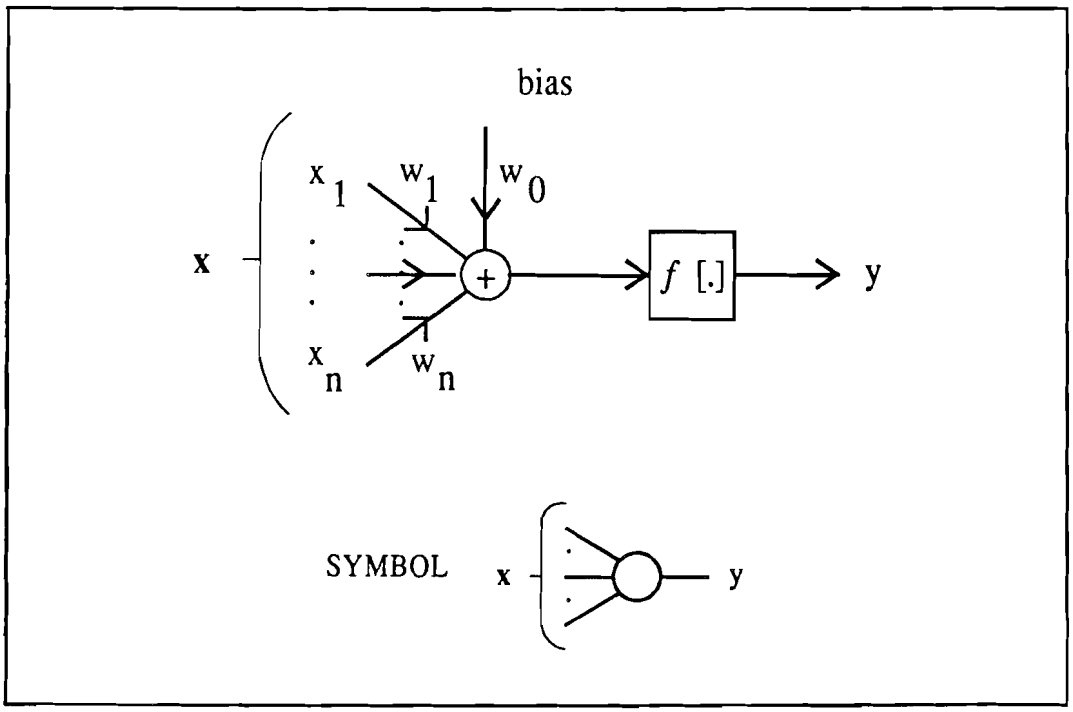


Figure 2.1 Structure of a processing element

A processing element has one or more inputs x_i , which it receives from processing elements or from the outside world. However it has one output, which can be connected to other processing elements or can act as an output of the neural network. Every input to the processing element, x_i, is weighted by a weightfactor w_i and summed with a weighted bias value. The output of the processing element is obtained by passing the weighted inputs through a processing function $f[.]$. The relation between the inputvector **x** and the outputscalar y of a processing element can be described by formula 2.1.

$$y = f[\sum_{i=1}^{n} (x_i.w_i) + bias.w_0] = f[(\mathbf{x}.\mathbf{w}) + bias.w_0] \qquad (2.1)$$

Commonly used processing functions are the linear-, ramp-, step- and sigmoid functions, figure 2.2.

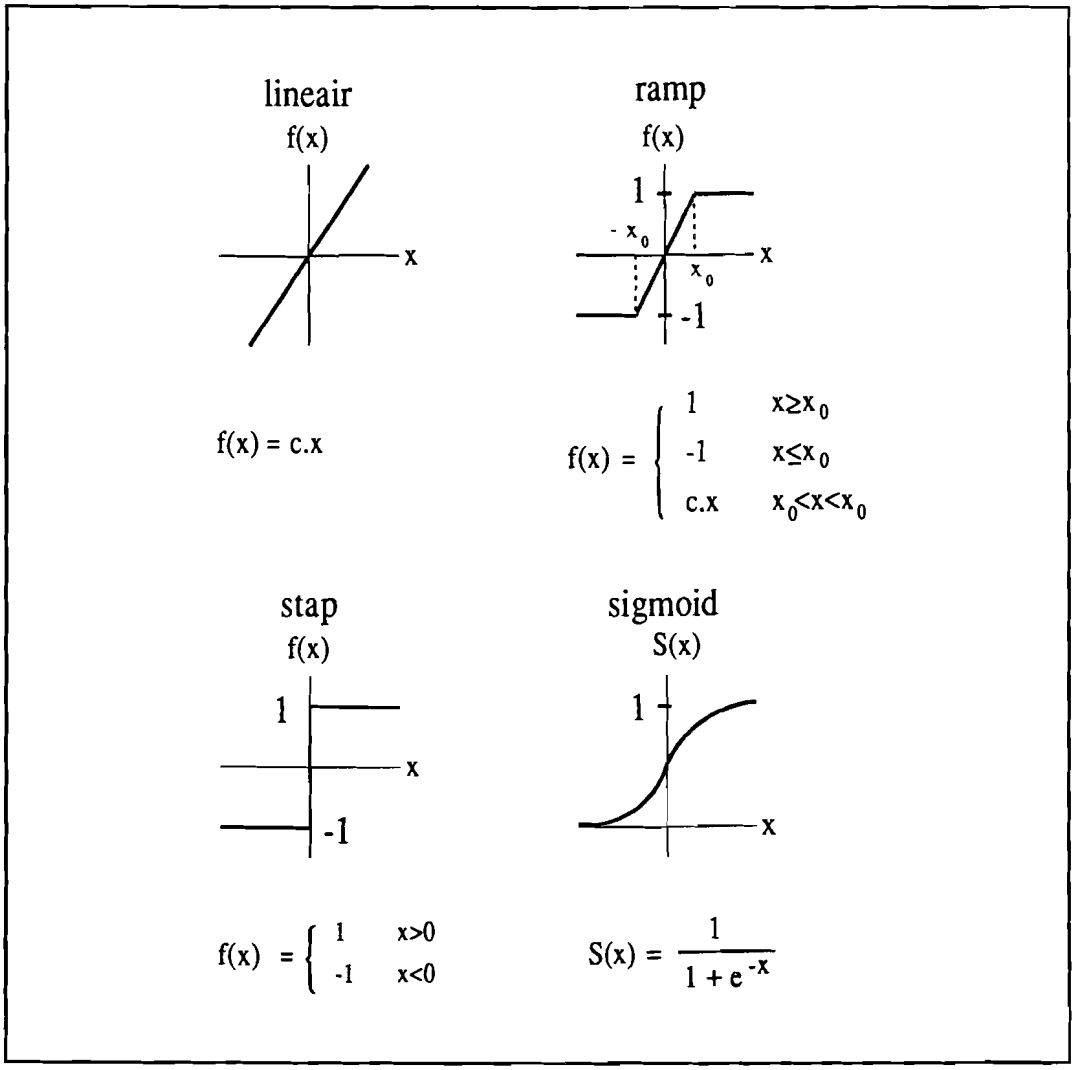


Figure 2.2 Commonly used transfer functions

A nonlinear processing function gives the neural network the ability to represent nonlinear relationships. As will be explained in section 2.3, the learnalgorithm used imposes a restriction on the processing function, the first derivative has to exist. Sigmoid functions are of prime interest because they exhibit a linear, nonlinear and a saturation behaviour which is useful in nonlinear system identification. Because of this advantage a sigmoid function is frequently used. Three kind of sigmoid functions are applied (figure 2.3):

$$S(x)=\frac{1}{1+e^{-x}} \tag{2.2}$$

$$S(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.3}$$

$$S(x)=\frac{1-e^{-x}}{1+e^{-x}} \tag{2.4}$$

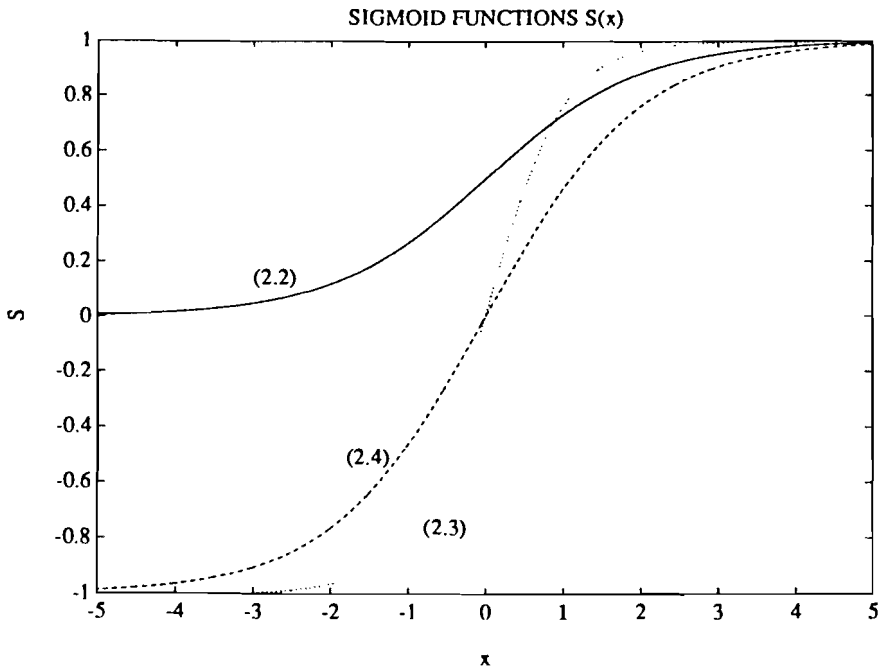


Figure 2.3 Three sigmoid functions

As can be seen from the processing element model, the bias term affects the coordinate space of the processing function, which enables the network to characterize the structure of the non-linearity.

The network topology is formed by dividing the processing elements up into groups (layers) and making connections between the processing elements. There are three kind of layers:

- input layer : A collection of processing elements who receive their inputsignals from the outside world.
- output layer: A collection of processing elements who send their output to the outside world.
- hidden layer: Every layer between the input- and output layer.

To create a neural network structure the processing elements are connected by connections. As can be seen in figure 2.1, each connection has a weight value which defines the 'strength' of the connection. In general there are three types of connections to distinguish.

1. Intra-layer connections: connections between processing elements from one layer.
2. Inter-layer connections: connections between processing elements from different layers. There are two inter-layer connections: feedforward connections, connections in one direction, and feedback connections, connections in both directions (when a loop exists in the network).
3. Recurrent connections : connections which connect the output of a processing element with its own input.

In figure 2.4 the different connection types are given.

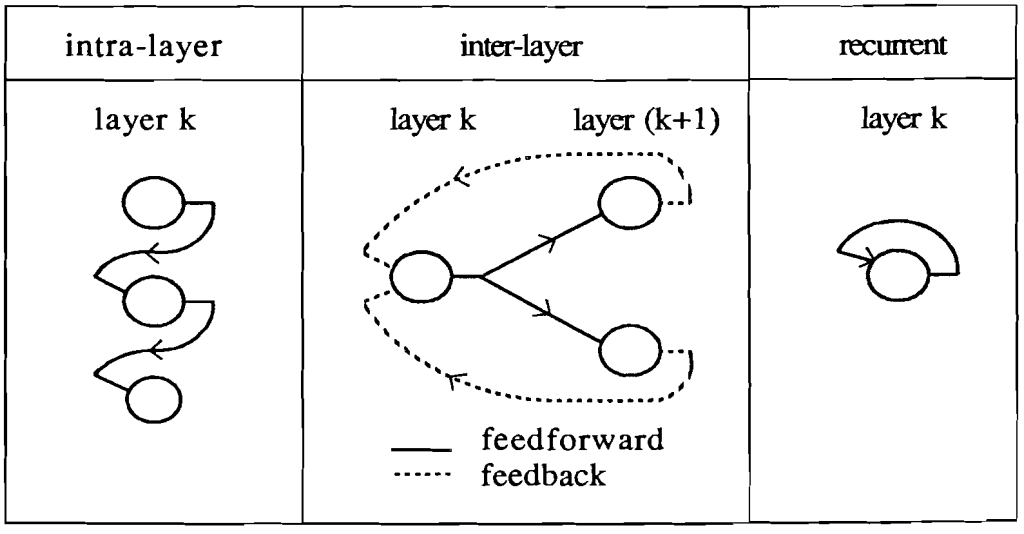


Figure 2.4 Different network connection types

2.2 Multilayer neural networks

For the use of process modelling, a neural network is needed that learns input-output relations, from input-output data pairs of the process. A neural network which is able to learn static input-output relationships is the so called multilayer neural network (MNN). Figure 2.5 gives the topology of an MNN.

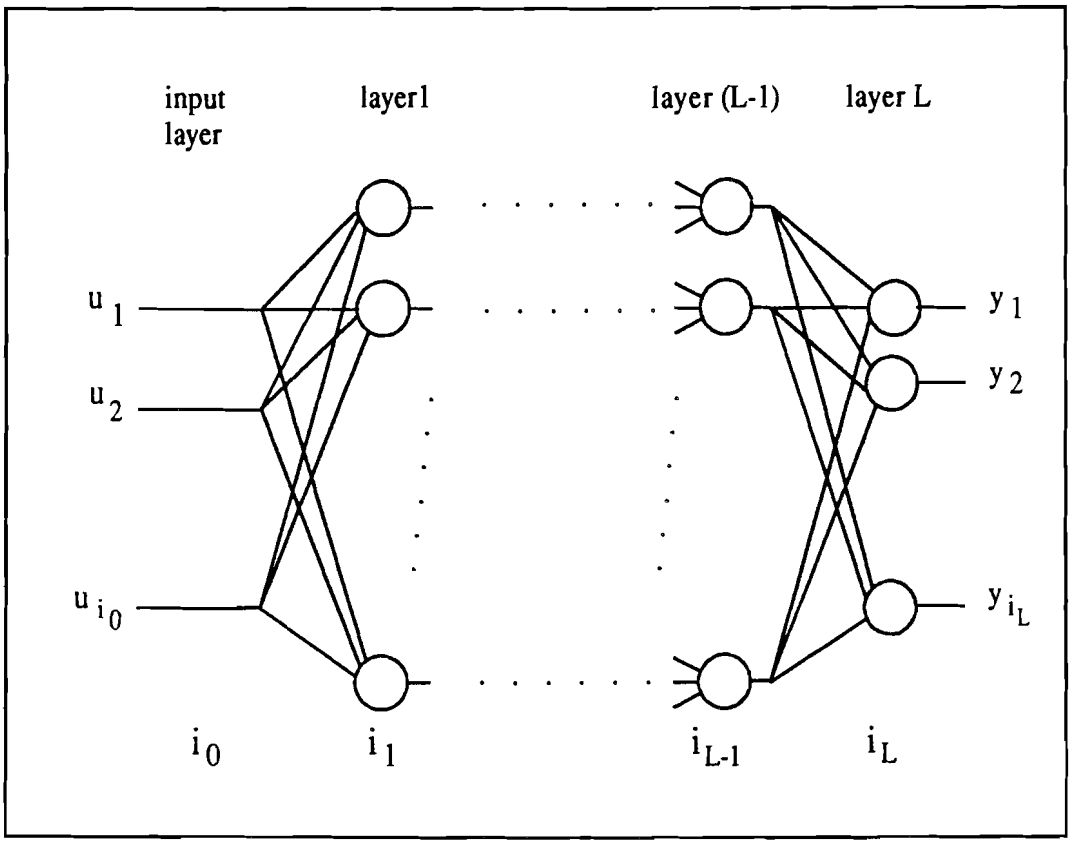


Figure 2.5 The topology of a multilayer neural network

The MNN has the subjoined properties.

- The input layer distributes the inputs to the network over the first hidden layer. While the processing elements of the input layer have only one input and a linear processing function, they are fed through elements, and not real processing elements.
- The network consists only of feedforward inter-layer connections, which only connects processing elements of layer l with processing elements of layer (l+1).
- The network is completely interconnected, this means that all possible connection, provided allowed, are made.
- All processing elements are represented by circles. The processing elements of the hidden layers have the same processing function. Typically a sigmoid function is chosen.

- For identification applications it's preferable that the output of a neural network model can describe a wide range. However the output of a processing element is limited by the range of the used processing function. Depending on the kind of sigmoid function in the range (0,1) or (-1,1). To meet the requirement of a wide output range, the processing functions of the output layer processing elements are chosen to be linear.
- The number of processing elements in layer l are represented by the parameter i_l. So the number of inputs is i_0 and the number of outputs is i_L.
- Every layer l is extended with i_l weights according to the bias term.

The MNN can also be denoted in a block diagram, figure 2.6. Each layer of the network is represented by two blocks, a *W*-block representing the weights associated with the connections coming into this layer and a *S*-block containing the processing functions going with the processing elements of this layer.

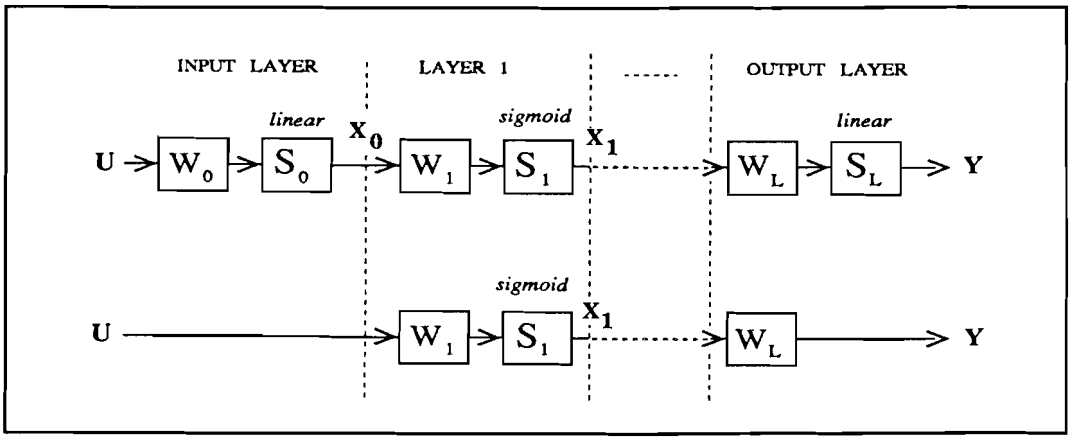


Figure 2.6 Block diagram representation of a multilayer neural network

The matrices *W* and *S* are of the form.

$$W^l = \begin{bmatrix} w^l_{11} & w^l_{21} & . & . & w^l_{i_{l-1}1} \\ . & . & . & . & . \\ . & . & . & . & . \\ w^l_{1i_l} & w^l_{2i_l} & . & . & w^l_{i_{l-1}i_l} \end{bmatrix} \qquad S^l = \begin{bmatrix} S^l_1[.] & 0 & . & . & 0 \\ 0 & S^l_2[.] & . & . & 0 \\ . & . & . & . & . \\ 0 & 0 & . & . & S^l_{i_l}[.] \end{bmatrix} \qquad (2.5)$$

where l hidden layer number

i_{l-1} number of processing elements in layer (l-1)

i_l number of processing elements in layer l

w_{pq} weightfactor of connection between processing element p and q of respectively layer l and (l-1).

According to the properties of an MNN, the W and S matrices of the input- and output layer are respectively: input layer $W = S = \mathbf{I}$

output layer $S = \mathbf{I}$

So the input-output relation can be given by equation 2.6.

$$\begin{aligned} &\text{one layer} && x_l = S_l[W_l . x_{l-1}] \\ &\text{total network} && y = S_L[W_L S_{L-1}[W_{L-1} S_{L-2}[. \; . \; S_1[W_1 S_0[W_0 . u]] . \; .]]] \\ & && y = W_L S_{L-1}[. \; . \; S_1[W_1 . u] . \; .] \end{aligned} \tag{2.6}$$

For ease of notation a neural network will be notated as follow:

$$N_{i_0 i_1 \ldots i_{L-1} i_L}$$

where L number of layers (without input layer)

L-1 number of hidden layers

i_0 number of inputs

i_L number of outputs

$i_1..i_{L-1}$ number of processing elements in respectively layer 1..L-1

A formula can be derived which calculates the number of parameters Q in the MNN.

$$Q = \sum_{l=1}^{L} (i_{l-1} + 1) i_l \tag{2.7}$$

From formula 2.7 it can be seen that multilayer neural networks with the same number of processing elements, but with a different number of hidden layers, contain not an equal number of parameters. For example an $N_{1\,10\,1}$ MNN has 31 parameters and an $N_{1\,5\,5\,1}$ has 46 parameters.

Finally the value of neural networks can be judged by a theorem of Funahashi [16], which is posed here without prove.

Theorem

f[.] non-constant, bounded and monotonically increasing function (sigmoid function)

K compact set on $\mathbb{R}^{i0}$

$\Psi_s(u_1 \ldots u_{i0})$ real valued continuous function on K

For any $\epsilon > 0$, there exists an integer i_1 and the real constants w_{r0}, w_{rq} and w_{sr} with ($q=1...i_0$; $r=1...i_1$; $s=1...i_2$) such that:

$$\psi_s(u_1 \ldots u_{i_0}) = \sum_{r=1}^{i_1} w_{sr} f[\sum_{q=1}^{i_0} w_{rq} u_q - w_{r0}] \tag{2.8}$$

satisfies

$$\max \mid \Psi_s(u_1 \ldots u_{i_0}) - \psi_s(u_1 \ldots u_{i_0}) \mid < \epsilon \tag{2.9}$$

This results in the MNN structure given in figure 2.7.

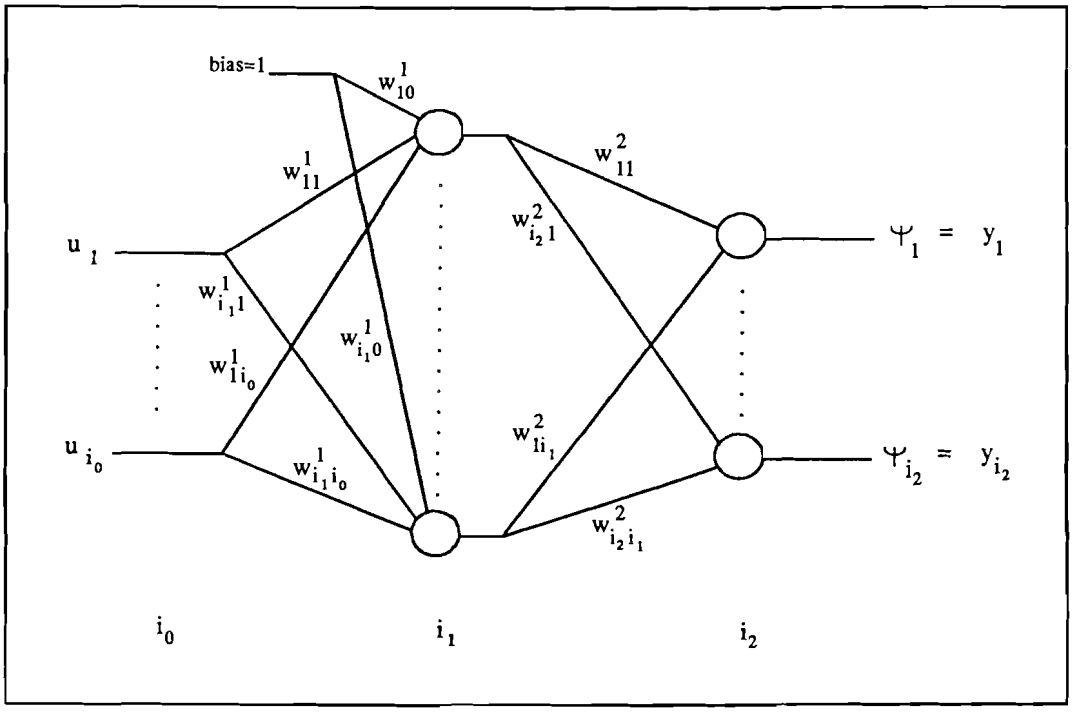


Figure 2.7 MNN stated by Funahashi

The theorem states that any continuous MNN with one hidden layer with a fixed continuous sigmoidal nonlinearity and a linear output layer, can approximate any continuous function arbitrarily well on a compact set. Although this theorem does not give any insight into the size of the hidden layer (the number of processing elements i_1) and the learnalgorithm to obtain this MNN, it does show that the fundamental structure of an MNN is such that it can model any continuous nonlinear function. It should be noticed that it is often advantageous to use more than one hidden layer, because it can lead to an MNN containing less processing elements.

2.3 Learning algorithm for a multilayer neural network

A neural network can be seen as a model with the weights as the model parameters. To perform some specific task, the weights of the network have to be set somehow to get a desired input-output behaviour. A process, involving a so called learning algorithm, takes place, in which the weights of the network are adjusted to reproduce the desired behaviour. The learning algorithm adjusts the parameters (weights) of the neural network based on a given set of input-output pairs: learn patterns. If the weights of the network are considered as elements of a parameter set W, the learning process involves the determination of the parameter set $\mathbf{W}^*$ which optimizes a performance function J based on the output error. Backpropagation is the most commonly used method for this purpose in the static case (for multilayer neural networks). The performance function which has to be minimized is the least squares criterion.

$$J = \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}\frac{1}{2}[\hat{y}_p(k) - y_p(k)]^2 \tag{2.10}$$

where $\hat{y}_p(k)$ model output number p
$y_p(k)$ system output number p
N batch size
i_L number of outputs

The backpropagation algorithm minimizes the performance function according to the well known steepest descent method. The parameter set W of the neural network is, after some learnpatterns, modified in such a way that counts $J(\mathbf{W}^{new}) < J(\mathbf{W}^{old})$. This means that W has to be moved in the direction $-\nabla_W J(\mathbf{W})$. Formula 2.11 gives the mathematical form of the steepest descent method.

$$W^{new} = W^{old} - \alpha \nabla_W J \tag{2.11}$$

where $\mathbf{W}^{new}$ new weight parameters
$\mathbf{W}^{old}$ old weight parameters
α learnvelocity factor
$\nabla_W J$ gradient of J w.r.t. W

The modification of each separate weight is according to the subjoined expressions.
The derivation of this expressions is restored in appendix A.

$$w_{rq}^{new} = w_{rq}^{old} - \alpha \frac{\delta J}{\delta w_{rq}} \qquad \text{with} \quad \frac{\delta J}{\delta w_{rq}} = \frac{1}{N} \sum_{k=1}^{N} \delta_r(k) x_q(k) \tag{2.12}$$

where q processing element of layer l-1
r processing element of layer l
x_q the output of processing element q
w_{rq} weight value of the connection from processing element q to r

When r is a processing element of a hidden layer:

$$\delta_r(k) = f'[net_r(k)] \sum_{s=1}^{i_{l+1}} \delta_s(k) w_{sr} \tag{2.13}$$

where s processing element of layer l+1

When r is a processing element of the output layer:

$$\delta_r(k) = [\hat{y}_r(k) - y_r(k)]\, f'[net_r(k)] \qquad \text{with} \quad f'[net_r(k)] = 1 \tag{2.14}$$

The parameter α is the learnvelocity factor, and has a value in the range [0,1]. When α is chosen too large, the algorithm will not converge to a minimum, but become instable. When, however, α is too small, the learning time will be very large. This means that there's an optimum value for α, where the algorithm is just stable and the convergence time minimal. According to experience α must be chosen in the range [0.1,0.25].

In the backpropagation algorithm there are two distinct phases, a forward phase and a backward phase. During the forward phase the inputs **u**(k) are propagated forward through the network to produce the network outputvector $\hat{\mathbf{y}}$(k). The neural network outputvector $\hat{\mathbf{y}}$(k) is compared to the desired outputvector **y**(k), and an errorsignal δ_p(k) is calculated for every processing element of the outputlayer (formula 2.14). During the backward phase the errorsignal δ_p(k) is propagated backward through the network to calculate an errorsignal for every processing element separately, according to the recurrent relation 2.13. With this error signal and expression 2.12 all weights can be modified.

As can be seen from expression 2.13 the first derivative of the processing function has to exist. Together with the ability to approximate nonlinear relationships, the sigmoid

function is the most appropriate processing function.

When the steepest descent method is used there is no guarantee that the algorithm will find a global minimum. The initial value and the properties of the error surface defines what kind of minimum will be found. The backpropagation error surface is described by the function $J=J(\mathbf{W})$ in the (Q+1)-dimensional space. Herewith counts $J \in \mathbb{R}$ and $\mathbf{W} \in \mathbb{R}_Q$, with Q networkconnections (weights). Until now very little is known about the property of backpropagation error surfaces. Three aspects are mentioned in [13].

1. Experiments have showed that a lot of backpropagation error surfaces have large areas with a small slope. In this areas weights have to change considerable to give a significant reduction of the performance function.
2. Backpropagation error surfaces have a lot of global minima. There is not a unique minimum by the fact that there exists more than one parameter set that describes the same input-output behaviour (no canonical form).
3. It has been proven [13] that backpropagation error surfaces possess local minima. About the place and number of local minima in proportion to global minima is nothing known yet.

In view of the first property of backpropagation error surfaces, the conventional backpropagation algorithm has the disadvantageous property that the converge time increases when the slope $\nabla_W J(\mathbf{W})$ becomes smaller. To speed up the backpropagation algorithm there are some improved algorithms developed.

- The 'jump every time' version of the backpropagation algorithm modifies the weights after every learnpattern. In this case the batch size N is one, so formula 2.12 passes into the subjoined expression.

$$w_{rq}^{new} = w_{rq}^{old} - \alpha \, \delta_r(k) \, y_q(k) \tag{2.15}$$

When the gradient is calculated over N samples and the parameters are not adjusted every sample, the gradient is the superposition of the gradient over every single sample. In formula 2.15, however, the weights are updated after every sample, so the next gradient will be calculated in another 'point', thus not the same result after N samples (as with formula 2.12) is obtained. Hopefully a faster convergence will appear. Figure 2.8 shows the difference between the standard backpropagation algorithm and the 'jump every time' version.

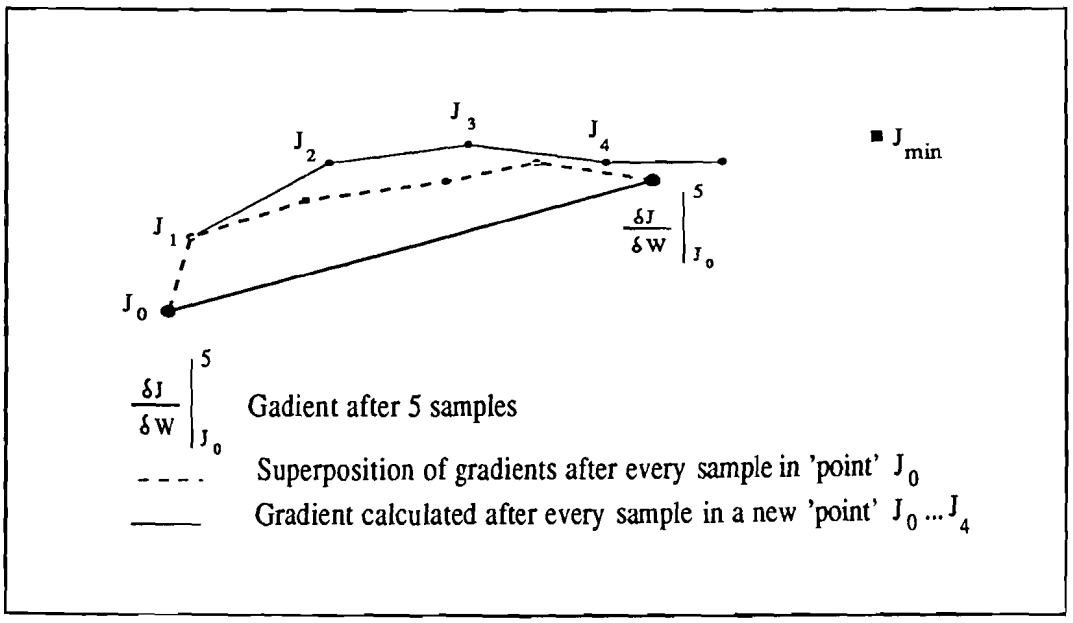


Figure 2.8 The steepest descent method when N=1 and N>1

- The momentum version of the backpropagation algorithm includes a momentumterm. This term is proportional to the last weight modification, expression 2.16.

$$W^{l+1} = W^{l} + \alpha \nabla_{W} J(W) + \beta [W^{l} - W^{l-1}] \tag{2.16}$$

- Making the learnvelocity factor α variable is another way of speed-up the algorithm. This kind of algorithms is called Self Adapting Backpropagation SAB. In [10] some heuristics are discussed.
- A better way to improve the learning rate is the use of Newton-like algorithms instead of gradient techniques. The weight modification is then according to expression 2.17.

$$W^{new} = W^{old} - \left[\frac{\delta^2 J(W)}{\delta W \delta W^T}\right]^{-1} \frac{\delta J(W)}{\delta W} \tag{2.17}$$

The Newton algorithm converges quadratically (i.e. very fast) as the initial values of the parameters are close enough to the minimum value of J. In a lot of cases this requirement is not met, so a combination of steepest descent and Newton will give the solution, the Marquardt algorithm 2.18.

$$W^{new} = W^{old} - \left[\lambda I + \frac{\delta^2 J(W)}{\delta W \delta W^T}\right]^{-1} \frac{\delta J(W)}{\delta W} \tag{2.18}$$

In the beginning λ is large, so 2.18 will pass into 2.11. Further on in the learning process λ is small, so 2.18 will pass into 2.17. The steepest descent method is used to get near the minimum, and Newton is used to get at the minimum in high speed.
The second derivative of J to W will be calculated as follows:

$$\frac{\delta^2 J(W)}{\delta W \delta W^T} = \frac{\delta}{\delta W^T}\frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L} e_p(k)\frac{e_p(k)}{\delta W}$$

$$= \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}\left[\frac{\delta e_p(k)}{\delta W^T}\frac{\delta e_p(k)}{\delta W} + e_p(k)\frac{\delta^2 e_p(k)}{\delta W \delta W^T}\right] \qquad (2.19)$$

neglected

The last term can be neglected when J is close to its minimum since $e_p(k)$ is small (Gauss-Newton method). The advantage is that the remaining term always leads to a positive definite matrix and consequently no difficulties in the inversion process. Despite of the extra calculation time, which is necessary to calculate the inverse of the second derivative, there is a faster convergence in a lot of practical applications.

2.4 Recurrent multilayer neural networks

The MNN performs a nonlinear mapping between the inputs and outputs. Dynamics are not included within their structure. Processes which have to be identified do have dynamics. Dynamics can be included within the processing element, by extending the processing function with a first-order lowpass filter with the time constant as an extra parameter [3]. A more obvious solution, but not as elegant, uses time histories of data. The input of the MNN will be extended with time histories of inputs as well as outputs. An MNN which arises is called a recurrent multilayer neural network, RMNN. Recurrent multilayer neural networks can be constructed of only two basic operations.

1. delay network
2. nonlinear operator N[.], a multilayer neural network

Figure 2.9 shows the most simple form of an RMNN.

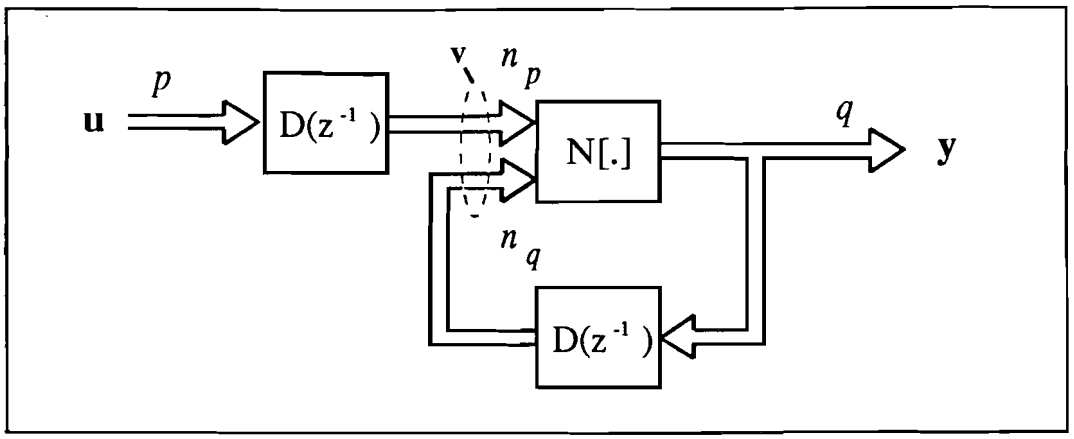


Figure 2.9 The structure of a recurrent multilayer neural network

If we have p inputs, q outputs and the number of relevant delays is n for both inputs and outputs, then the delay operators are given by:

$$D_1(z^{-1}) = \begin{bmatrix} I_p \\ z^{-1}I_p \\ z^{-2}I_p \\ . \\ . \\ . \\ z^{-n}I_p \end{bmatrix} \qquad D_2(z^{-1}) = \begin{bmatrix} z^{-1}I_q \\ z^{-2}I_q \\ . \\ . \\ . \\ z^{-n}I_p \end{bmatrix}$$

The modification of the weights of the RMNN can not be done by the backpropagation algorithm. As can be seen in appendix A for the calculation of $\nabla_W J$ the knowledge of $\nabla_W y$ is enough. According to formula 2.20, an extra term in the calculation of $\nabla_W J$ will occur because of the feedback of past outputs.

$$\begin{aligned} \frac{dy}{dW} &= \frac{\delta N[v]}{\delta W} + \frac{\delta N[v]}{\delta v} \cdot \frac{\delta v}{\delta y} \cdot \frac{dy}{dW} \\ &= \frac{\delta N[v]}{\delta W} + \frac{\delta N[v]}{\delta v} . D(z^{-1}) . \frac{dy}{dW} \end{aligned} \tag{2.20}$$

The last term in 2.19 contains the past derivatives of y to W. Since this term makes the calculation of the gradient $\nabla_W J$ complex (time consuming and no guaranteed convergence) it will be neglected in the beginning. So the learn algorithm will be conform to the back propagation algorithm. Practical applications and a special identification procedure have shown (section 3.2) that good results will be obtained.

3 SYSTEM IDENTIFICATION

The ability of neural networks to approximate nonlinear relationships makes them prime candidate for the use in nonlinear system identification. To model nonlinear systems, nonlinear system descriptions have to be given. Four nonlinear models and a universal model are discussed. To identify systems two identification models are discussed, the equation error model and the output error model. Because a universal system description will be used, which includes a linear ARMA model besides a neural network model, an extended learnalgorithm will be derived. In conclusion an identification protocol for nonlinear system identification will be discussed.

In this chapter only the SISO case is discussed. It is obvious that the same discussion can be stated for the MIMO case. Some restraints are imposed on the nonlinear systems to be identified:
- The system has to be BIBO stable (Bounded Input Bounded Output).
- A tight bound should be available for the inputs and outputs to the system in operation.

3.1 System description

To identify nonlinear systems it is necessary to define nonlinear models, whose parameters have to be estimated to represent the system. Narendra and Parthasaraty [12] introduce four nonlinear models, derived from the linear ARMA (Auto Regressive Moving Average) model. The well known ARMA model is represented by the subjoined linear difference equation.

$$y(k) = A(z^{-1})y(k) + [b_0 + B(z^{-1})]u(k) \tag{3.1}$$

with $A(z^{-1}) = a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{n_y} z^{-n_y}$
$B(z^{-1}) = b_1 z^{-1} + b_2 z^{-2} + \ldots + b_{n_u} z^{-n_u}$
and z^{-1} a delay operator

Based on this model, nonlinearities can be introduced. Depending on the nonlinearity, four models can be derived, represented by difference equation 3.2 up to 3.5.

Model 1

$$y_p(k)=A(z^{-1})y_p(k)+g[u(k),u(k-1),\ldots,u(k-n_u)] \tag{3.2}$$

with $A(z^{-1})=a_1z^{-1}+a_2z^{-2}+\ldots+a_{n_y}z^{-n_y}$

Model 2

$$y_p(k)=f[y_p(k-1),y_p(k-2),\ldots,y_p(k-n_y)]+[b_0+B(z^{-1})]\,u(k) \tag{3.3}$$

with $B(z^{-1})=b_1z^{-1}+b_2z^{-2}+\ldots+b_{n_u}z^{-n_u}$

Model 3

$$y_p(k)=f[y_p(k-1),y_p(k-2),\ldots,y_p(k-n_y)]+g[u(k),u(k-1),\ldots,u(k-n_u)] \tag{3.4}$$

Model 4

$$y_p(k)=f[y_p(k-1),y_p(k-2),\ldots,y_p(k-n_y);u(k),u(k-1),\ldots,u(k-n_u)] \tag{3.5}$$

where f[.] and g[.] are general nonlinear functions

In all four models the output of the system at time k depends on its past outputs, the input at time k and past inputs. The dependence on the input at time k and past inputs is nonlinear in model 1, while the dependence on the past outputs is nonlinear in model 2. In model 3 the nonlinear dependence on present and past inputs and outputs is separable. It is evident that model 4, in which the output is a nonlinear function of past inputs and outputs, subsumes model 1 to 3. Model 4 is, however, analytically the least tractable while it includes the most model parameters which have to be estimated. So it is advantageous to use one of the other models if any a priori information about the nonlinearities in the system is available. It can reduce the number of model parameters considerably. It is obvious that the nonlinear functions f and g of the different system descriptions will be modelled by a multilayer neural network.

The block diagram representations of the various models are displayed in figure 3.1.

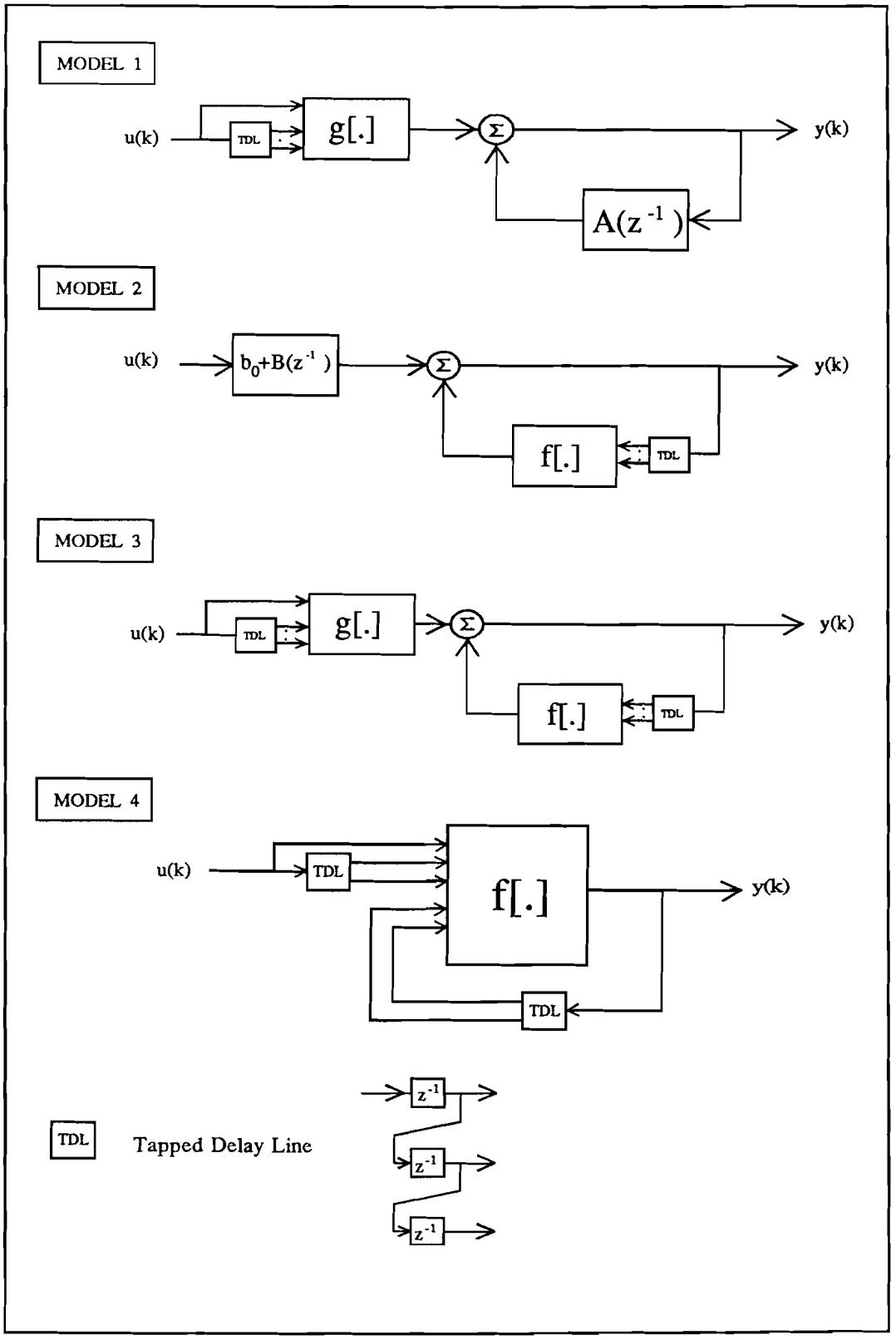


Figure 3.1 Block diagram representation of model 1 to 4

In a lot of practical situations much is known about the linear part of a system, obtained from linear identifications. This a priori knowledge can not be used completely in one of the models mentioned before. Here a fifth model will be introduced which is capable to use this a priori information and which contains all four models given by Narendra and Parthasarathy. This universal model can be described by the difference equation

$$y(k)=A(z^{-1})y(k)+[b_0+B(z^{-1})]u(k)+f[u,y]+g[u,y] \tag{3.6}$$

with $u=[u(k),u(k-1),\ldots,u(k-n_u)]$

$y=[y(k-1),y(k-2),\ldots,y(k-n_y)]$

$A(z^{-1})=a_1z^{-1}+a_2z^{-2}+\ldots+a_{n_y}z^{-n_y}$

$B(z^{-1})=b_1z^{-1}+b_2z^{-2}+\ldots+b_{n_u}z^{-n_u}$

Two separate nonlinear networks have to be used in order to incorporate the previous model 3. A schematic representation is drawn in figure 3.2.

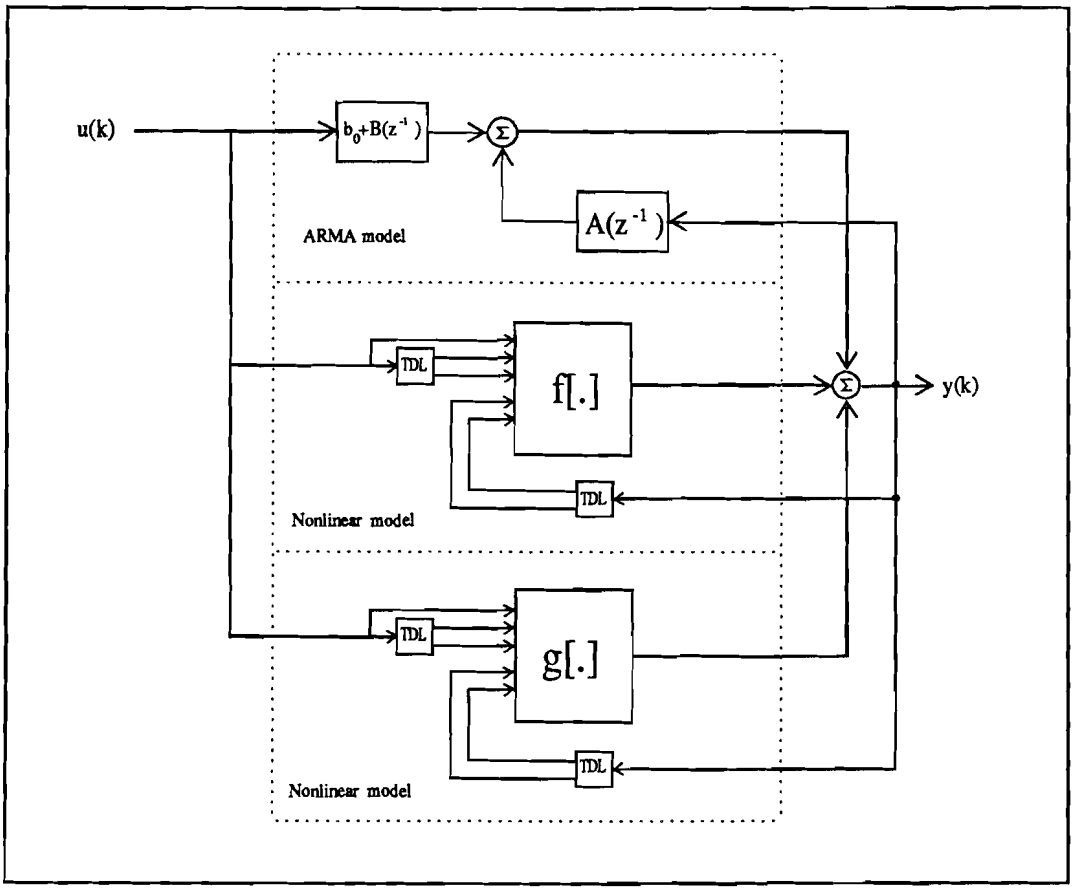


Figure 3.2 Block diagram representation of model 5

The advantages of this model over the other models are:

- There is no need to write different identification algorithms for each model. One algorithm can handle all models.
- It is possible to use a priori information from linear identification techniques directly. One can use ARMA parameters, obtained from earlier linear identifications, to initialize the ARMA parameters of model 5. So the nonlinear identification procedure estimates the neural network parameters and the ARMA parameters simultaneously, but the ARMA parameters are not expected to change much, because they have good starting values. This makes the identification procedure much faster.
- It is also possible to identify linear systems. This is not as effective as the existing identification algorithms due the fact that a steepest descent method is used.
- When the ARMA parameters are a priori known, something can be said about the amount of nonlinearities in the system. By fixing the ARMA parameters, the identification procedure will only estimate the nonlinear part.

3.2 Identification models

For neural network identification the same identification models are used as for conventional identification techniques. These are equation error model and output error model, also known as the series-parallel model and the parallel model. Both models are discussed.

Output error model

The blockscheme of the output error identification model is given by figure 3.3.

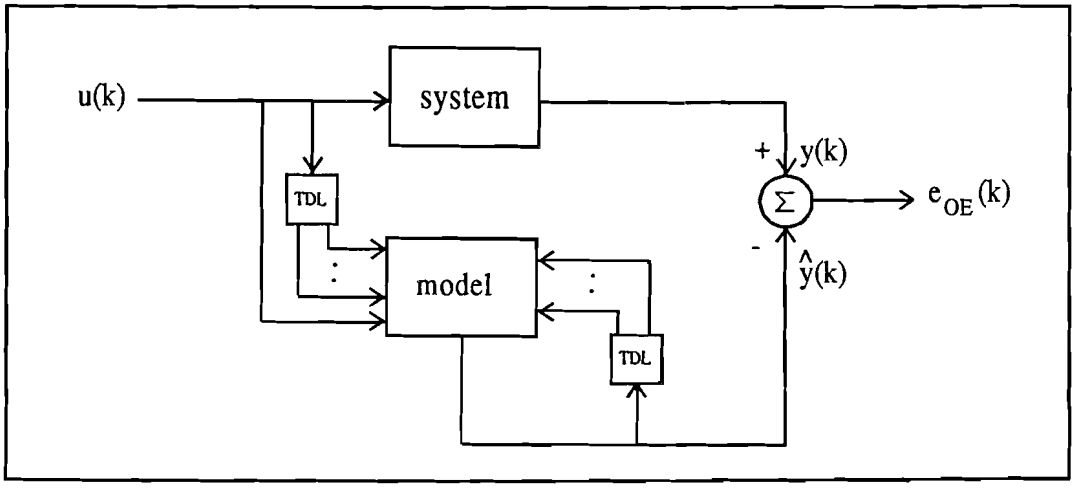


Figure 3.3 Output error identification model

As can be seen the inputs to the model are the system inputs and the past model outputs. Formula 3.7 gives an expression for the output error $e_{OE}(k)$.

$$\begin{aligned} \text{system} \quad & y(k) = A(z^{-1})y(k) + [b_0 + B(z^{-1})]u(k) + f(u,y) + g(u,y) \\ \text{model} \quad & \hat{y}(k) = \hat{A}(z^{-1})\hat{y}(k) + [\hat{b}_0 + \hat{B}(z^{-1})]u(k) + N_1[u,\hat{y}] + N_2[u,\hat{y}] \\ \text{with} \quad & e_{OE}(k) = y(k) - \hat{y}(k) \end{aligned} \tag{3.7}$$

As can be seen from figure 3.3 the model used is a recurrent multilayer neural network. The disadvantage of this model is the learnalgorithm which has to be used to train the network. The algorithm becomes very time consuming because of the recurrent relationship and there's no guarantee that the output error ŷ(k)-y(k) will tend to zero (section 2.4).

Equation error model

The blockscheme of the equation error identification model is displayed in figure 3.4.

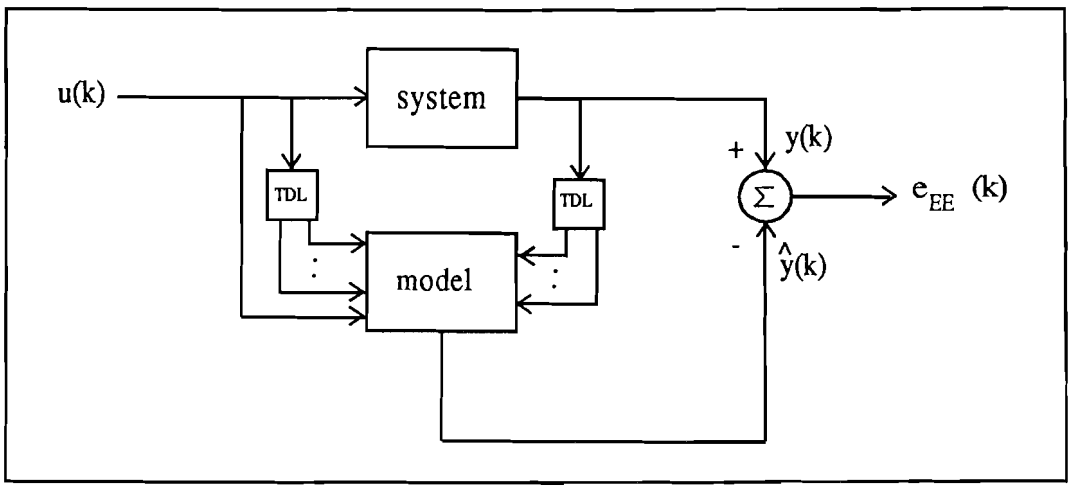


Figure 3.4 Equation error identification model

An expression for the equation error $e_{EE}(k)$ can be derived.

$$\begin{aligned} \text{system} \quad & y(k) = A(z^{-1})y(k) + [b_0 + B(z^{-1})]u(k) + f(u,y) + g(u,y) \\ \text{model} \quad & \hat{y}(k) = \hat{A}(z^{-1})y(k) + [\hat{b}_0 + \hat{B}(z^{-1})]u(k) + N_1[u,y] + N_2[u,y] \\ \text{with} \quad & e_{EE}(k) = y(k) - \hat{y}(k) \end{aligned} \tag{3.8}$$

In contrast to the output error model, the inputs to the model are the system inputs as well as the system outputs. While there's no feedback loop in the model (no recurrent relations) a multilayer neural network can easily be trained. The training time will reduce considerably when the backpropagation algorithm is used. Since the system is supposed to be BIBO stable and the inputs and outputs are bounded, all signals in the identification model are bounded. Though the model would not give a good simulation, because the estimation is fully based upon past outputs (the inputs are 'masked').However, the prediction is very good. To overcome the problem of bad simulation and the difficult learning of the output error model, the subjoined identification procedure will be followed. The identification is started with the equation error identification model. When no improvement is obtained, the identification model is switched to the output error identification model. In this way the convergence problem of the output error model will hopefully be solved, and a good simulation model will be found.

3.3 The extended backpropagation algorithm

For the estimation of the parameters of model 5, it is necessary to estimate the parameters of the ARMA model and the two neural network models simultaneously. For simplicity, the ARMA parameters are also estimated with the steepest descent method. The parameter set θ consists of the ARMA parameters A, B and the parameters of the multilayer neural networks $\mathbf{W}_1$ and $\mathbf{W}_2$. The gradient $\nabla_\theta J$ can be calculated as follows:

The performance function J is:

$$J = \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}\frac{1}{2}[\hat{y}_p(k) - y_p(k)]^2$$

$$\frac{\delta J}{\delta \theta} = \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}[\hat{y}_p(k) - y_p(k)]\frac{\delta \hat{y}_p(k)}{\delta \theta}$$

For the SISO case counts $i_L = 1$, so:

$$\frac{\delta J}{\delta \theta} = \frac{1}{N}\sum_{k=1}^{N}[\hat{y}(k) - y(k)]\frac{\delta \hat{y}(k)}{\delta \theta} \qquad (3.9)$$

Output error model

$$\text{system} \quad y(k) = A(z^{-1})y(k) + [b_0 + B(z^{-1})]u(k) + f(u,y) + g(u,y)$$

$$\text{model} \quad \hat{y}(k) = \hat{A}(z^{-1})\hat{y}(k) + [\hat{b}_0 + \hat{B}(z^{-1})]u(k) + N_1[u,\hat{y}] + N_2[u,\hat{y}]$$

$$\text{with} \quad e_{OE}(k) = y(k) - \hat{y}(k)$$

$$\theta \in \hat{A}(z^{-1}) \quad \frac{\delta J}{\delta \hat{a}_i} = \frac{1}{N}\sum_{k=1}^{N} \hat{y}(k-i).e_{OE}(k) \quad \text{with} \quad 1 \le i \le n_y$$

$$\theta \in \hat{B}(z^{-1}) \quad \frac{\delta J}{\delta \hat{b}_j} = \frac{1}{N}\sum_{k=1}^{N} u(k-j).e_{OE}(k) \quad \text{with} \quad 0 \le j \le n_u \qquad (3.10)$$

$\theta \in W_1$ recurrent backpropagation algorithm

$\theta \in W_2$ recurrent backpropagation algorithm

This is only the first approximation, because ŷ(k-i) is also a function of θ.

Equation error model

$$\text{system} \quad y(k) = A(z^{-1})y(k) + [b_0 + B(z^{-1})]u(k) + f(u,y) + g(u,y)$$

$$\text{model} \quad \hat{y}(k) = \hat{A}(z^{-1})y(k) + [\hat{b}_0 + \hat{B}(z^{-1})]u(k) + N_1[u,y] + N_2[u,y]$$

$$\text{with} \quad e_{EE}(k) = y(k) - \hat{y}(k)$$

$$\theta \in \hat{A}(z^{-1}) \quad \frac{\delta J}{\delta \hat{a}_i} = \frac{1}{N}\sum_{k=1}^{N} y(k-i).e_{EE}(k) \quad \text{with} \quad 1 \le i \le n_y$$

$$\theta \in \hat{B}(z^{-1}) \quad \frac{\delta J}{\delta \hat{b}_j} = \frac{1}{N}\sum_{k=1}^{N} u(k-j).e_{EE}(k) \quad \text{with} \quad 0 \le j \le n_u \qquad (3.11)$$

$\theta \in W_1$ backpropagation algorithm

$\theta \in W_2$ backpropagation algorithm

To define the structure of model 5, a so called connection matrix will be introduced. Figure 3.5 gives the place of the connection matrix in the model.

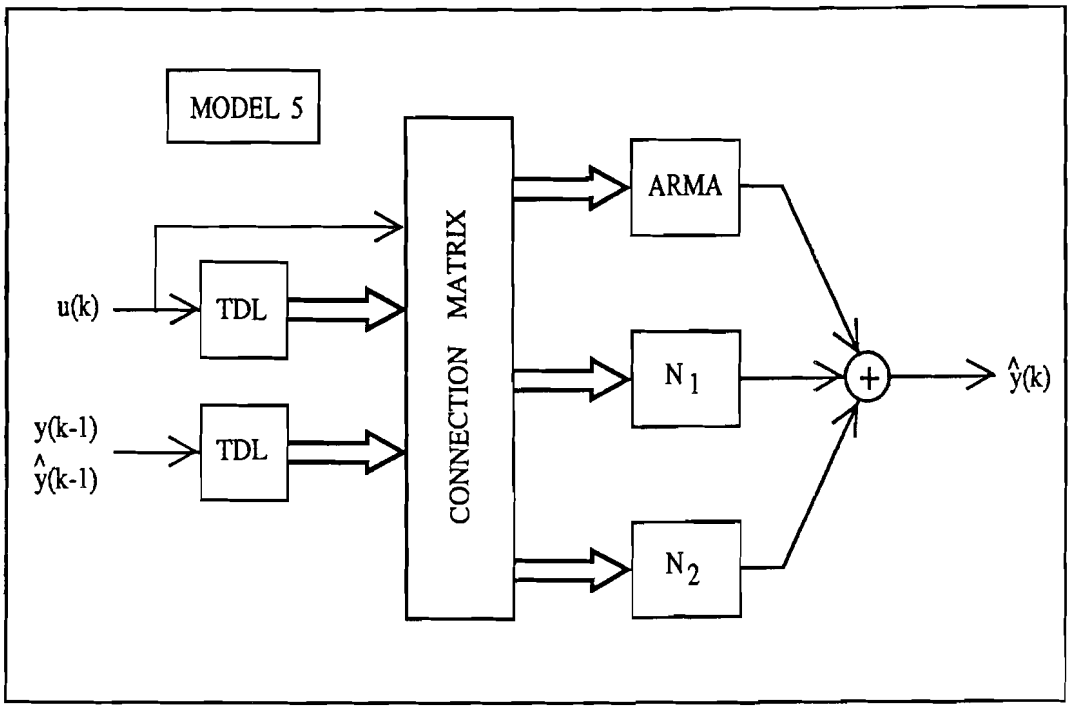


Figure 3.5 Structure definition of model 5 by a connection matrix

The connection matrix consists of three rows and a variable number of columns. Every row defines the structure of a 'sub-model'. Row 1 defines the structure of the ARMA model, row 2 and 3 the structure of respectively neural network 1 and 2. The number of columns is defined by the process order. First connection matrices for sample k to (k-q), where q is the system order, are defined.

$$\begin{matrix} u_k & & u_{k-1} & & u_{k-2} & & & & u_{k-q} \\ \downarrow & & \downarrow & & \downarrow & & & & \downarrow \end{matrix}$$

$$\begin{bmatrix} . & 0 \\ . & 0 \\ . & 0 \end{bmatrix} z^0 + \begin{bmatrix} . & . \\ . & . \\ . & . \end{bmatrix} z^{-1} + \begin{bmatrix} . & . \\ . & . \\ . & . \end{bmatrix} z^{-2} + \ldots\ldots\ldots + \begin{bmatrix} . & . \\ . & . \\ . & . \end{bmatrix} z^{-q}$$

$$\begin{matrix} \uparrow & & \uparrow & & \uparrow & & & & \uparrow \\ y_k & & y_{k-1} & & y_{k-2} & & & & y_{k-q} \end{matrix}$$

Each matrix represents one delay time. The connection matrix is formed by cascading these matrices (equation 3.12).

The connection matrix is:

$$\begin{bmatrix} . & 0 & . & . & . & . & & & . & . \\ . & 0 & . & . & . & . & \cdots\cdots\cdots & & . & . \\ . & 0 & . & . & . & . & & & . & . \end{bmatrix} \begin{matrix} \leftarrow \text{ARMA model} \\ \leftarrow \text{Neural model 1} \\ \leftarrow \text{Neural model 2} \end{matrix} \quad (3.12)$$

$z^0 \quad z^{-1} \quad z^{-2} \quad z^{-q}$

The connection matrix will be filled with ones and zeros. When the concerning input/output is an input of the 'sub-model' the corresponding matrix entry is one. When the entry is zero, the corresponding parameter will not be estimated (for example fixed delays). To clarify the described procedure an example is given.

example

system model:

$$y(k) = a_1 y(k-1) + a_2 y(k-2) + b_0 u(k) + N_1[y(k-1), y(k-2)] + N_2[y(k-1), u(k), u(k-1)]$$

inputs: ARMA model : y(k-1),y(k-2),u(k)
Neural network 1: y(k-1),y(k-2)
Neural network 2: y(k-1),u(k),u(k-1)

So the connection matrix will be

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow \text{ARMA model} \\ \leftarrow \text{Neural model 1} \\ \leftarrow \text{Neural model 2} \end{matrix}$$

$u_k \quad u_{k-1} \quad y_{k-1} \quad y_{k-2}$

As can be seen from this example some properties of the connection matrix are:

- The number of ones per row define the number of inputs to the 'sub-model'.
- The system order is equal to the number of columns divided by two minus one.
- The second column of the connection matrix is filled with zeros, while this column represents the output at time k.

3.4 The identification protocol

As with linear techniques there are three different phases to distinguish, a preparation phase, a training phase and a validation phase. A schematic overview of the total identification protocol is restored in figure 3.6.

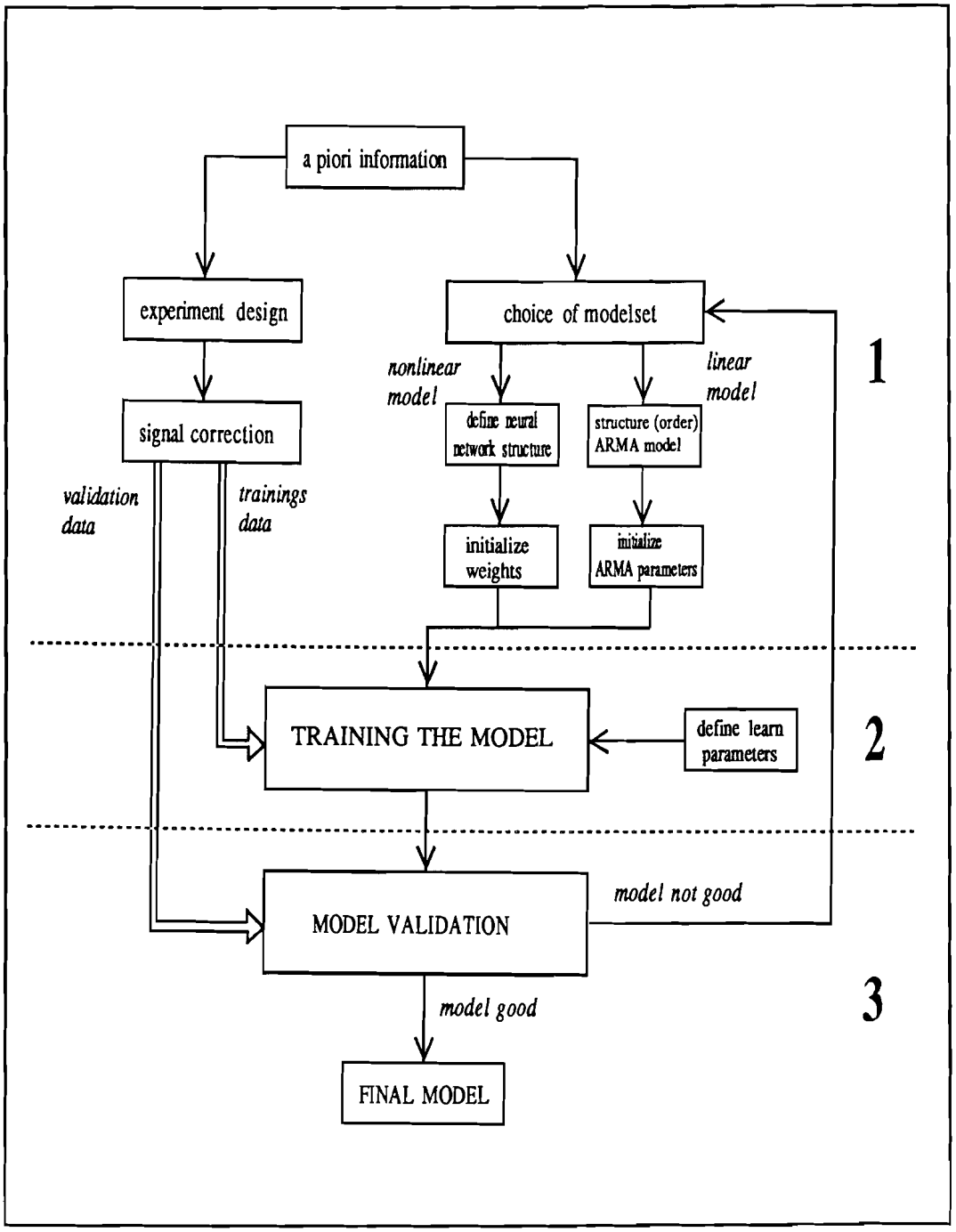


Figure 3.6 Schematic overview of the identification protocol

1. Preparation phase

In the preparation phase following aspects can be seen.

- Experimental design
  The input to the system need to be a noise-like signal, because otherwise the neural network tunes to the inputsignal in stead of the input-output relation. Random, uniformly distributed, white noise signals are preferable. Especially a PRBNS input (Pseudo Random Binary Noise Sequence), which is often used in linear identification techniques, is not suitable. Because the input value jumps between two discrete values the scale between this values is supposed to be linear, which is of course not the case for nonlinear systems.
- Signal check and validation
  Before using measured data it's necessary to do some data correction.
  - Trend, spike and time delay elimination.
  - Offset elimination, inputscaling in the range [-1,1] and output scaling to increase accuracy.
- Choice of modelset
  The choice of the modelset is fully based upon a priori knowledge. This a priori knowledge can be related to the model order, the number of inputs and outputs, the nonlinearity and so on. The use of this information is of essential importance, because it can reduce the number of model parameters considerable.
- Neural network structure definition
  There are some neural network structure parameters which must be defined.
  - The number of neural network inputs and outputs is defined by the number of system inputs and outputs and by the model order. The model order defines the number of outputs which have to be fed back, so together with the system inputs the total number of neural network inputs are known.
  - The number of hidden layers and processing elements per hidden layer depend on the complexity of the input-output relation which has to be mapped.
  - The type of sigmoid that will be used.
- Initializing the neural network parameters (weights)
  When the weights start with equal values the network can never learn. All the error signals to the hidden processing elements remain identical and the neural network begin at a local minimum and remains there. This problem can be counteracted by initializing the weights with small random values ($-0.25 \leq w \leq 0.25$).
- Initializing the ARMA parameters
  When there is a priori information about the ARMA parameters, from other earlier identifications, they are initially set to this values. Otherwise they are set to zero.

2. Training phase

During the training phase the model parameters are estimated by the extended backpropagation algorithm. Before starting the learning algorithm some learn parameters must be set.

- The learnvelocity factor α
  A good choice of this parameter is necessary, a bad choice can lead to instability or slow convergence. The learnvelocity factor of the linear estimation and the neural network estimation can be set separately. This is necessary because α_{NEURAL} can be chosen higher than α_{ARMA}, without leading to instability.
- The batch size N
  Normally the batch size is chosen to be one, but when there are a lot of learnpatterns it's better to calculate the mean gradient over a number (N) of learnpatterns. To speed up the learnalgorithm when great batch sizes are used, α can be increased without convergence problems.
- The number of iterations
  In order to reduce the total number of data samples, the same data is successively presented to the network. The times the same dataset is presented is also a learn parameter and depends upon the length of the dataset. A total of hundred thousand learnvectors (= number of samples times the number of repetitions) is normally enough to train a neural network.
- The momentum factor β
  According to the literature [3,10,13,14,21], the momentumfactor β can speedup the algorithm without instability problems. While experiments showed that this factor did not speedup the algorithm, β will be chosen zero.
- ARMA parameters fixed or estimated
  This parameters defines whether the ARMA parameters are estimated or not. When they are a priori known they can be fixed during training.

3. Validation phase

The validation phase is the third and final phase in the identification protocol. In this phase will be investigated if the model meets the previously set requirements. The model is accepted if it satisfies these requirements. If the requirements are not met, the model structure, especially the neural network structure, is modified and the training phase will be passed through again. Possibly with different initial weights to escape local minima.

The performance of the model will be tested by a validation dataset. To get a reliable validation, the validation dataset may not be used for network training. Most validation techniques for neural networks criticize the quality by a visual inspection. The model

4 SIMULATION RESULTS

In this chapter some simulation results are shown. In order to compare the obtained results, systems are taken from the examples given by Narendra and Parthasarathy [12]. They give examples of each system description discussed in section 3.1. Next to it an example of the universal model, model 5 is given. The identification of all systems is according to the described identification procedure. The identification will be started with the equation error identification model (EEM). When there's no improvement obtained, the identification model is switched to the output error identification model (OEM) to achieve a good simulation model. The performance of the estimated models is studied by comparing the system output with the model output using the output error model. The validation signals were chosen to be the same as the test signals used by Narendra and Parthasarathy. The performance given in the examples is calculated during training, and defined by equation 4.1.

$$\text{performance} = \frac{\sum_{k=1}^{T} [y(k) - \hat{y}(k)]^2}{\sum_{k=1}^{T} y^2(k)} \tag{4.1}$$

where: T number of samples in the training data set
y(k) system output at sample k
$\hat{y}$(k) model output at sample k

All systems satisfy the previously mentioned requirements: - BIBO stable
- bounded input
- bounded output

The neural network parameters are initially set to small random values. When the systems contain ARMA parameters, they were initially set to zero. In all simulations the estimated model parameters after the EEM identification act as starting values of the OEM identification.

The simulation results are divided into two groups, reviewed in the next sections.
- Simulation results without noise.
- Simulation results with additive white noise

4.1 Simulation results of systems without noise

In this section, simulation results are shown when the output of the system is not disturbed by some kind of noise.

Simulation 1

The system to be identified is described by the difference equation

$$y(k)=0.3y(k-1)+0.6y(k-2)+f[u(k)] \tag{4.2}$$

with

$$f[u(k)]=u(k)^3+0.3u(k)^2-0.4u(k)$$

training data : random, uniformly distributed noise in the interval [-1.5, 1.5]
validation data :

$$u(k)=\begin{cases} \sin(2\pi k/250) & 0\le k\le 250 \\ 0.75\sin(2\pi k/250)+0.25\sin(2\pi k/25) & 250<k\le 500 \end{cases}$$

EEM

$$\hat{y}(k)=\hat{a}_1 y(k-1)+\hat{a}_2 y(k-2)+N_1[u(k)] \tag{4.3}$$

The neural network structure is: $N_{1\,5\,5\,1}$
The learn parameters and the estimated a-parameters are gathered in table 4.1.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	0.1	0.05
α_{ARMA}	0.01	0.01
$\hat{a}_1$ (a_1=0.3)	0.3042	0.3024
$\hat{a}_2$ (a_2=0.6)	0.5948	0.5974
performance	4.55e-03	2.28e-03

Table 4.1 EEM identification results of simulation 1

The validation results after the last run are shown in figure 4.1.

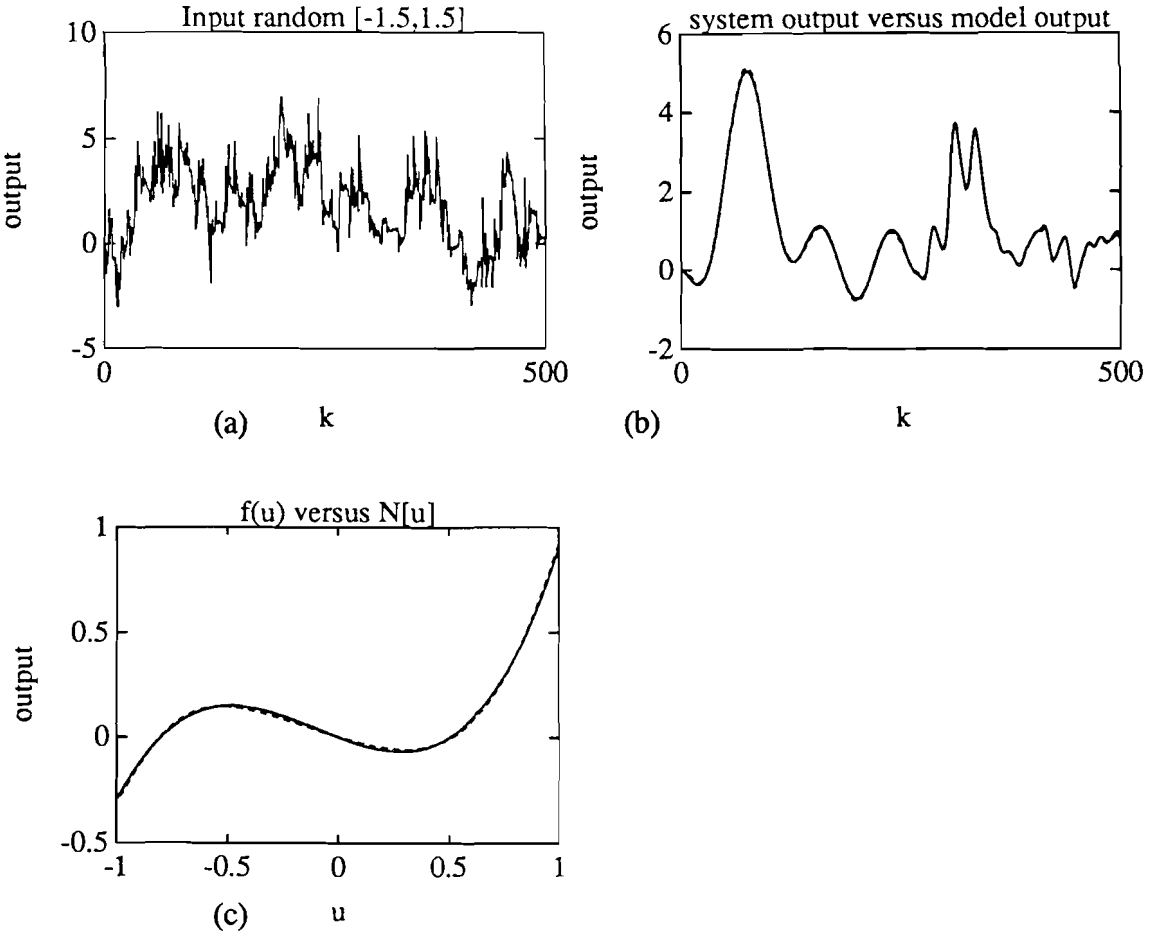


Figure 4.1 Model validation after EEM of simulation 1
(a) System output y(k) when the input is random [-1.5, 1.5]
(b) y(k) (solid) versus ŷ(k) (dashed)
(c) f[u] (solid) versus N_1[u] (dashed)

OEM

$$\hat{y}(k)=\hat{a}_1\hat{y}(k-1)+\hat{a}_2\hat{y}(k-2)+N_1[u(k)] \tag{4.4}$$

The neural network structure is: $N_{1\,5\,5\,1}$
The learn parameters and the estimated A parameters are gathered in table 4.2, and the validation results after the last run are shown in figure 4.2.

	run 1
learn patterns	500
iterations	50
batch size	1
α_{NEURAL}	0.001
α_{ARMA}	0.001
$\hat{a}_1$ (a_1=0.3)	0.3004
$\hat{a}_2$ (a_2=0.6)	0.5985
performance	2.2e-03

Table 4.2 OEM identification results of simulation 1

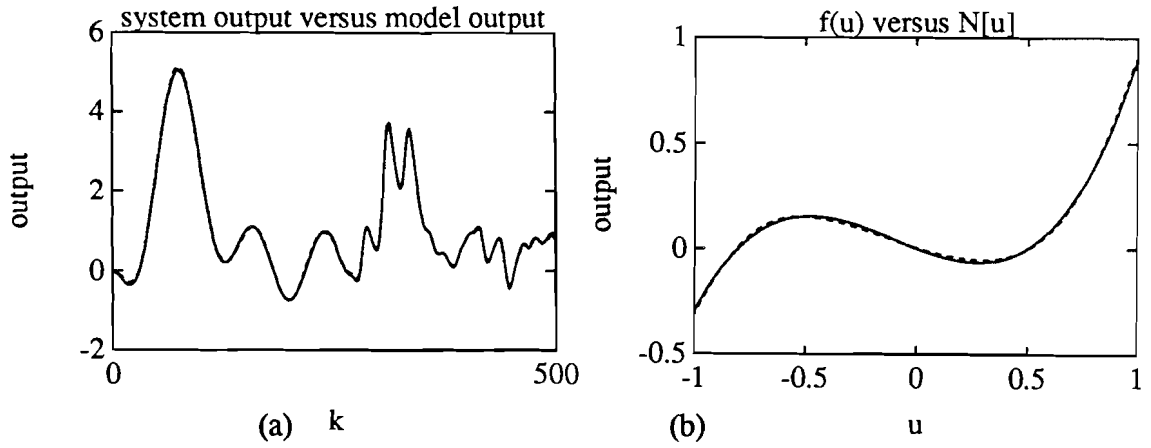


Figure 4.2 Model validation after OEM identification of simulation 1
(a) y(k) (solid) versus $\hat{y}$(k) (dashed)
(b) f[u] (solid) versus N_1[u] (dashed)

As can be seen from this simulation, the model approximates the system perfectly. While the linear part and the nonlinear part are fully separable (they are related to either the input or the output of the system), the AR parameters converge to the real values. Otherwise it is possible that the neural network models a piece of the linear part.

Simulation 2

The system to be identified is described by the difference equation

$$y(k)=f[y(k-1), y(k-2)]+u(k) \tag{4.5}$$

with

$$f[y(k-1),y(k-2)]=\frac{y(k-1)y(k-2)[y(k-1)+2.5]}{1+y(k-1)^2+y(k-2)^2}$$

training data : random, uniformly distributed noise in the interval [-2,2]
validation data :

$$u(k)=\sin(2\pi k/25)$$

EEM

$$\hat{y}(k)=N_1[y(k-1),y(k-2)]+b_0u(k) \tag{4.6}$$

The neural network structure is: $N_{2\,5\,5\,1}$
The learn parameters and the estimated b_0 parameter are gathered in table 4.3.

	run 1	run 2
learn patterns	1000	1000
iterations	25	25
batch size	1	1
α_{NEURAL}	0.05	0.01
α_{ARMA}	0.01	0.01
$\hat{b}_0$ (b_0=1)	1.004	1.003
performance	1.54e-02	1.14e-02

Table 4.3 EEM identification results of simulation 2

The validation results after the last run are shown in figure 4.3.

OEM

$$\hat{y}(k)=N_1[\hat{y}(k-1),\hat{y}(k-2)]+\hat{b}_0u(k) \tag{4.7}$$

Neural network structure $N_{2\,5\,5\,1}$
The learn parameters and the estimated b_0 parameter are gathered in table 4.4, and the simulation results after the last run are shown in figure 4.4.

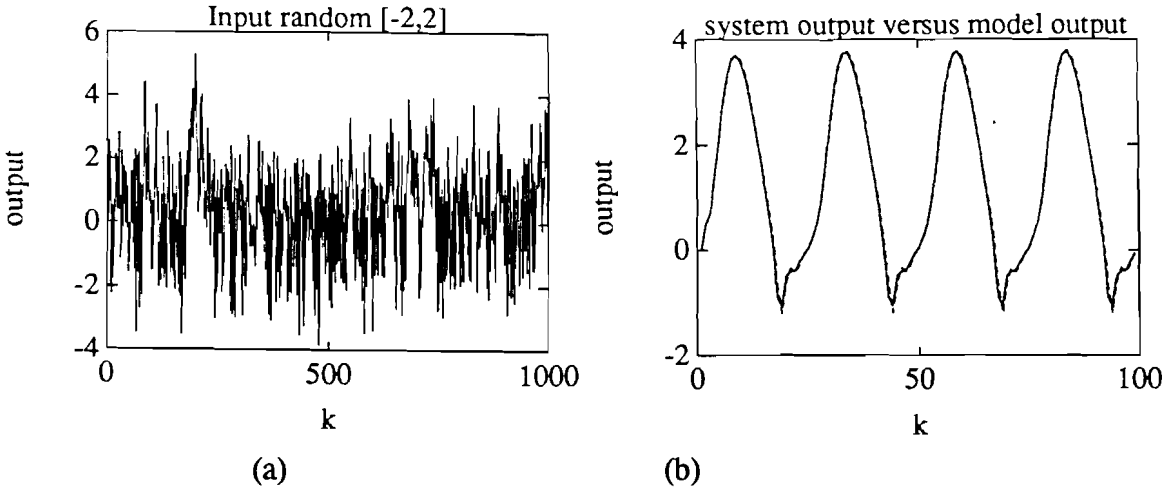


Figure 4.3 Model validation after EEM identification of simulation 2
(a) System output y(k) when the input is [-2,2]
(b) y(k) (solid) versus $\hat{y}(k)$ (dashed)

	run 1
learn patterns	1000
iterations	25
batch size	1
α_{NEURAL}	1.0e-04
α_{ARMA}	1.0e-04
$\hat{b}_0$ $(b_0=1)$	0.9971
performance	9.17e-03

Table 4.4 OEM identification results of simulation 2

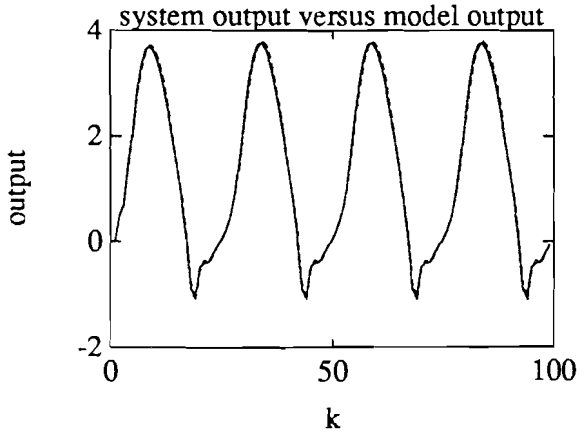


Figure 4.4 Model validation after OEM identification of simulation 2
y(k) (solid) versus $\hat{y}(k)$ (dashed)

As can be seen the model approximates the system very well. In contrast to simulation 1, a greater data set was used. There was no perfect fit when 500 samples were used, so a training set of 1000 samples were used for learning (values in between were not investigated). As in simulation 1, the linear and nonlinear part are fully separable so the b_0 parameter is estimated well.

Simulation 3

In this simulation the system is a model 3 type of the form:

$$y(k) = f[y(k-1)] + g[u(k)] \tag{4.8}$$

with

$$f[y(k-1)] = \frac{y(k-1)}{1+y(k-1)^2}$$

$$g[u(k)] = u(k)^3$$

Because the nonlinearities are separable, the identification model consists of two neural networks. Since the input was a random signal in the interval [-2,2], N_2 approximates g only over this interval. An input range of [-2,2] results in a variation of y over the interval [-10,10] so N_1 approximates f over the latter interval.

validation data :

$$u(k) = \sin(2\pi k/25) + \sin(2\pi k/10)$$

EEM

$$\hat{y}(k) = N_1[y(k-1)] + N_2[u(k)] \tag{4.9}$$

The neural network structures are: $N_1 = N_{1\,5\,5\,1}$ and $N_2 = N_{1\,5\,5\,1}$
The learn parameters are gathered in table 4.5 and the validation results after the last run are shown in figure 4.5.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	0.05	0.005
α_{ARMA}	--	--
performance	8.68e-04	2.00e-04

Table 4.5 EEM identification results of simulation 3

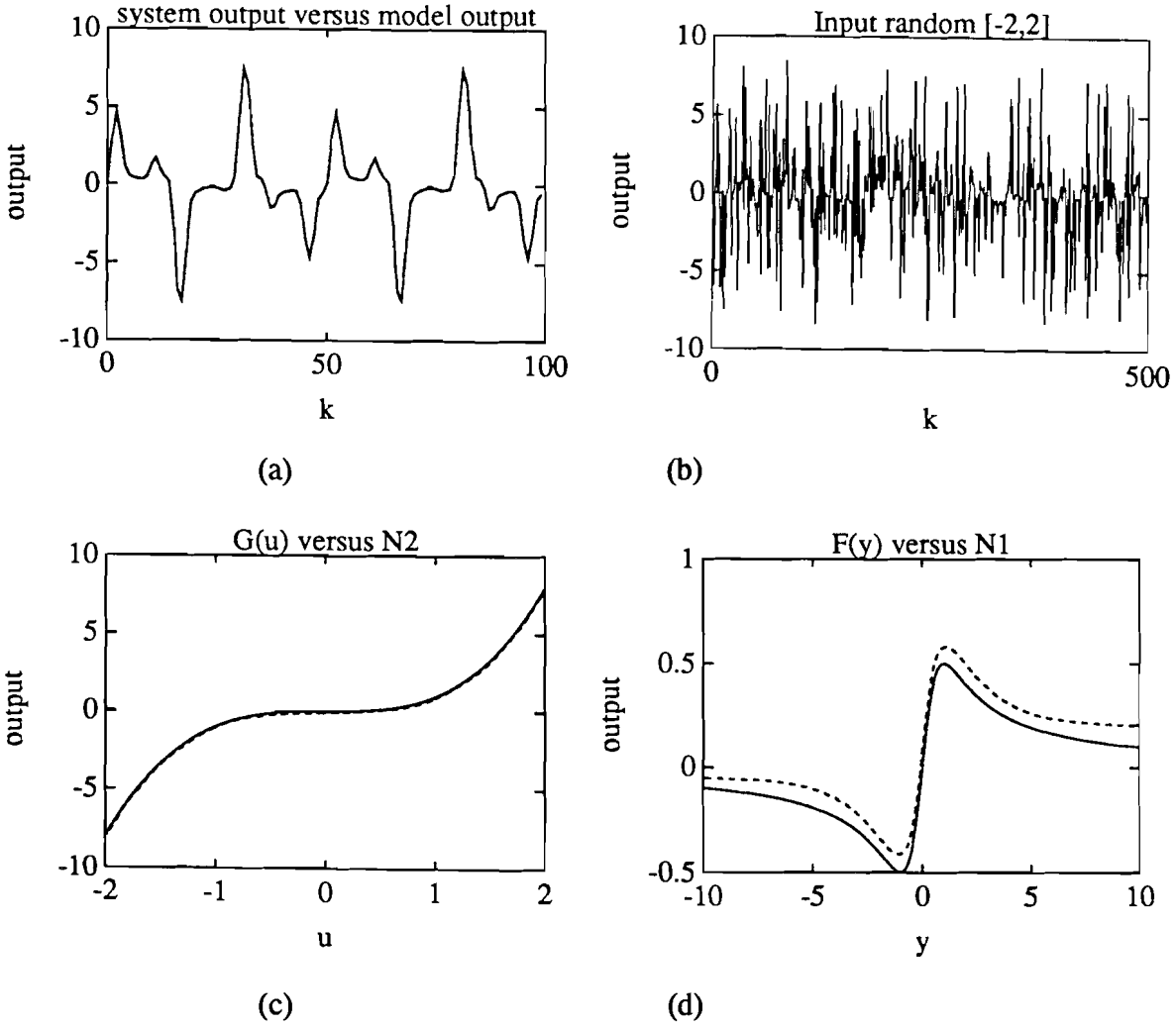


Figure 4.5 Model validation after EEM identification of simulation 3
(a) System output y(k) when the input is random [-2,2]
(b) y(k) (solid) versus $\hat{y}$(k) (dashed)
(c) g[u] (solid) versus N_2[u] (dashed)
(d) f[y] (solid) versus N_1[y] (dashed)

From figure 4.5 it can be seen that there is a mutual offset compensation (between N_1 and N_2). This offset can be clearly seen in figure 4.5(d) because g(u) is fairly dominant. Due to this compensation the overall model is good.

OEM

$$\hat{y}(k)=N_1[\hat{y}(k-1)]+N_2[u(k)] \tag{4.10}$$

The neural network structures are: $N_1=N_{1\,5\,5\,1}$ and $N_2=N_{1\,5\,5\,1}$
The learn parameters are gathered in table 4.6. The validation results after the last run are shown in figure 4.6.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	1.0e-03	1.0e-03
α_{ARMA}	--	--
performance	1.91e-04	1.90e-04

Table 4.6 OEM identification results of simulation 3

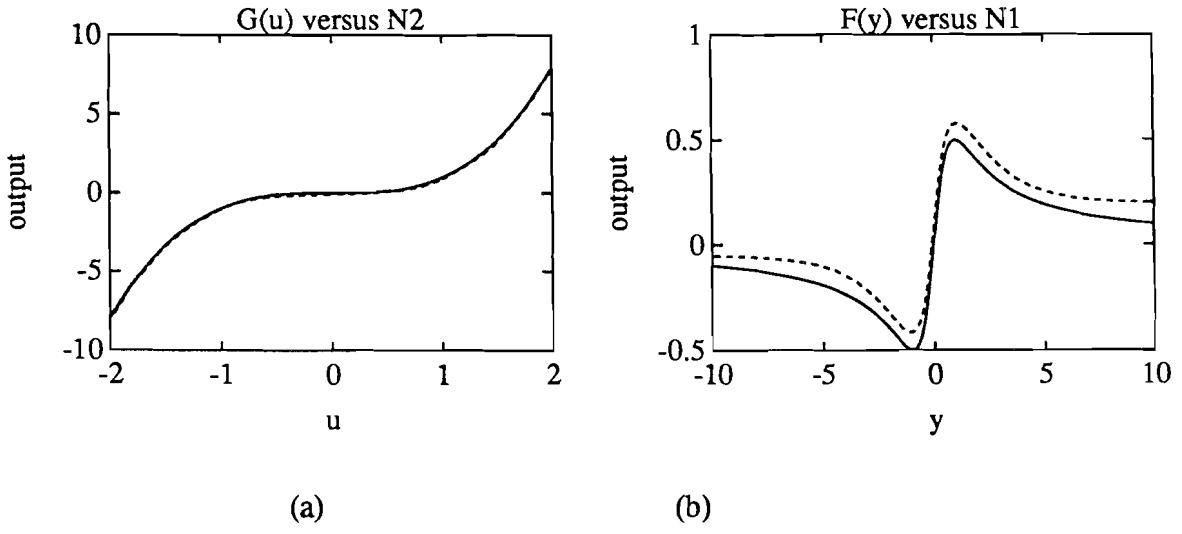
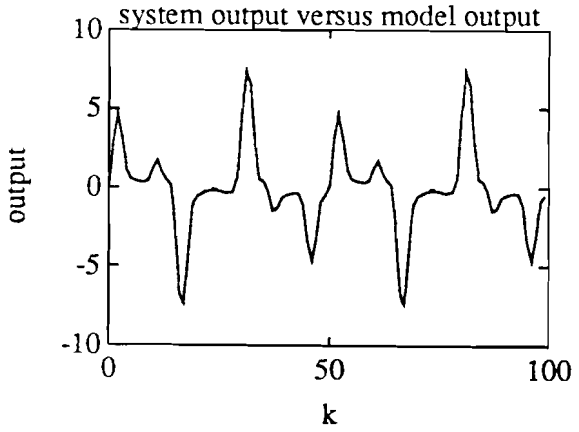


Figure 4.6 Model validation after OEM identification of simulation 3
(a) g[u] (solid) versus N_2[u] (dashed)
(b) f[y] (solid) versus N_1[y] (dashed)



(c)

Figure 4.6 Model validation after OEM identification of simulation 3
(c) y(k) (solid) versus $\hat{y}$(k) (dashed)

Simulation 4

The system to be identified is described by:

$$y(k)=f[y(k-1),y(k-2),y(k-3);u(k),u(k-1)] \tag{4.11}$$

with

$$f[x_1,x_2,x_3;x_4,x_5]=\frac{x_1x_2x_3x_5[x_3-1]+x_4}{1+x_2^2+x_3^2}$$

training data : random, uniformly distributed noise in the interval [-1,1]
validation data :

$$u(k)=\begin{cases}\sin(2\pi k/250) & 0\le k\le 500\\ 0.8\sin(2\pi k/250)+0.2\sin(2\pi k/25) & 500<k\le 800\end{cases}$$

EEM

$$\hat{y}(k)=N_1[y(k-1),y(k-2),y(k-3);u(k),u(k-1)] \tag{4.12}$$

The neural network structure is: $N_{5\,5\,5\,1}$
The learn parameters are gathered in table 4.7.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	0.1	0.1
α_{ARMA}	--	--
performance	8.01e-02	8.01e-02

Table 4.7 EEM identification results of simulation 4

The validation results after the last run are shown in figure 4.7.

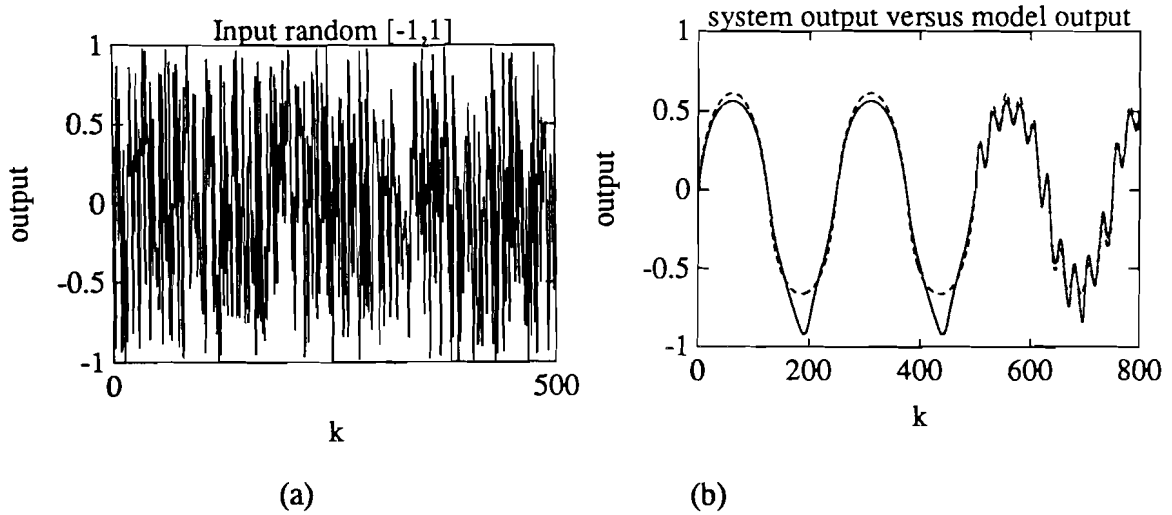


Figure 4.7 Model validation after EEM of simulation 4
(a) System output y(k) when the input is [-1,1]
(b) y(k) (solid) versus $\hat{y}$(k) (dashed)

OEM

$$\hat{y}(k)=N_1[\hat{y}(k-1),\hat{y}(k-2),\hat{y}(k-3);u(k),u(k-1)] \tag{4.13}$$

Neural network structure $N_{5\,5\,5\,1}$
The learn parameters are gathered in table 4.8.

	run 1
learn patterns	500
iterations	50
batch size	1
α_{NEURAL}	0.01
α_{ARMA}	--
performance	8.01e-02

Table 4.8 OEM identification results of simulation 4

The validation results after the last run are shown in figure 4.8.

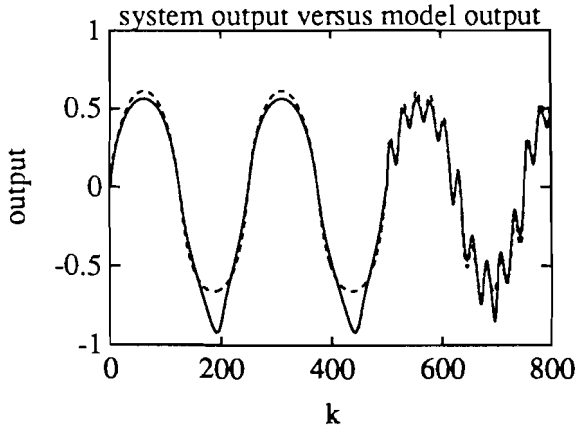


Figure 4.8 Model validation after OEM identification of simulation 4
y(k) (solid) versus $\hat{y}$(k) (dashed)

As can be seen the model can not 'follow' the system completely. The performance remains unchanged after the first run of the EEM identification (this can also be seen in the validation results, no improvement). It may be possible that the learning algorithm is stuck in a local minimum. Identifications, however, with other neural network structures and initialisations, showed no improvement. Maybe increase of the batch size or extra low frequency contents in the learning set may be of any help (see chapter 5 where initially same effects occur). A suggestion for further research is the use of the validation dataset as training set to see if this leads to improvement.

Simulation 5

The system to be identified is a single water vessel, figure 4.9.

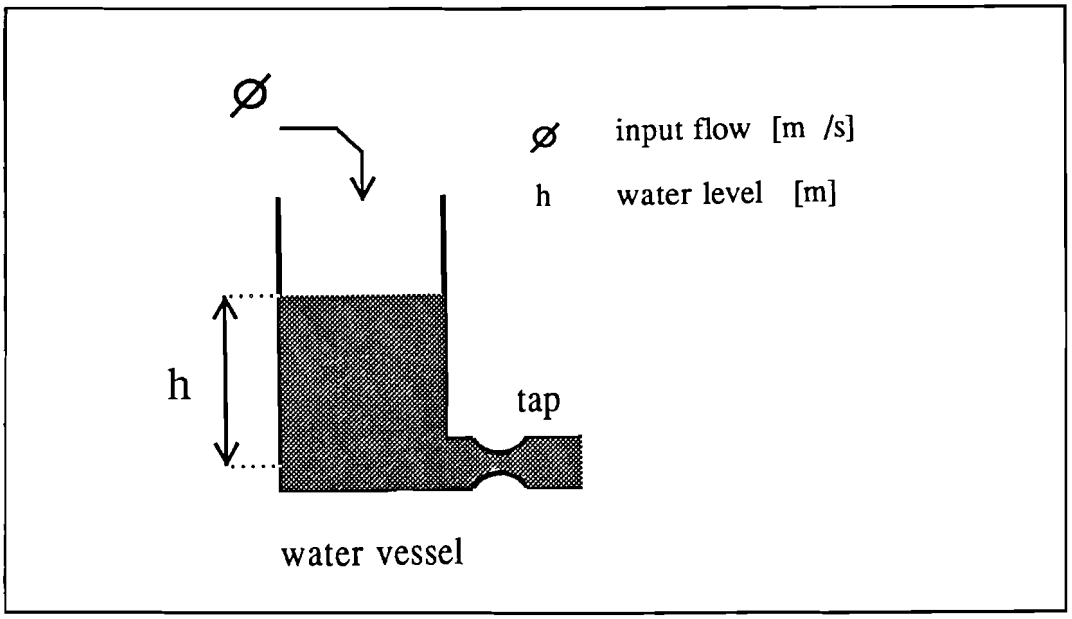


Figure 4.9 A single water vessel system

The input is the flow into the vessel (ϕ) and the output the water level in the vessel (h). The difference equation is derived in appendix B.

$$h(k) = a_1 h(k-1) + b_1 \phi(k-1) + f[h(k-1)] \tag{4.14}$$

with $a_1 = 1$
$b_1 = 1500$

The input and the output are scaled to obtain a higher accuracy.
input data : random, uniformly distributed signal [0, 3e-05] [m^3/s]
validation data : random, uniformly distributed signal [0, 3e-05] [m^3/s]

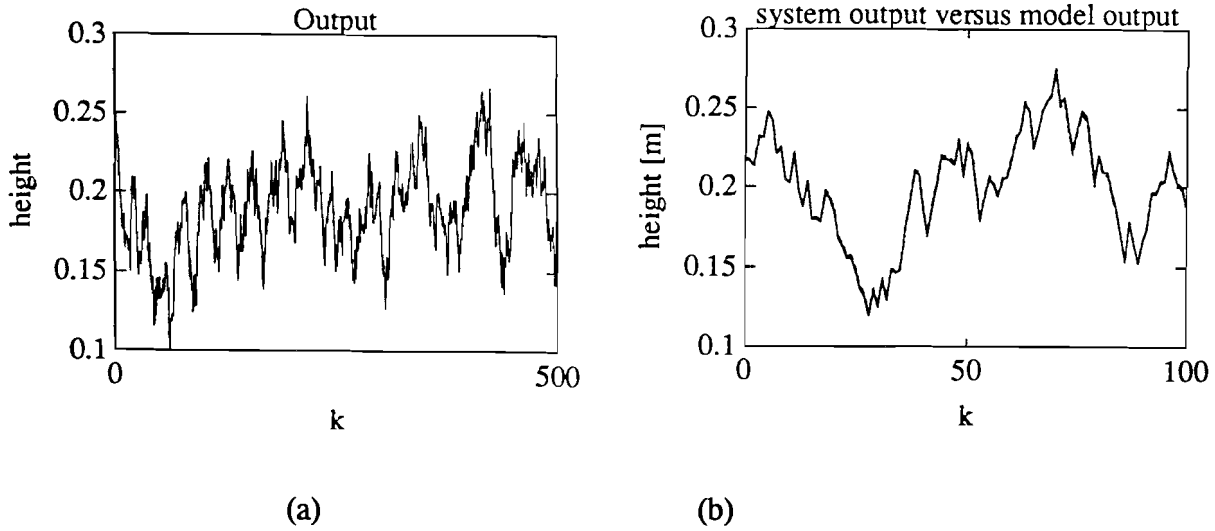
EEM

$$\hat{h}(k) = \hat{a}_1 h(k-1) + \hat{b}_1 \phi(k) + N_1[h(k-1)] \tag{4.15}$$

The neural network structure is: $N_{1\,5\,5\,1}$
The learn parameters and the estimated ARMA parameters are gathered in table 4.9. The validation after the last run is shown in figure 4.10.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	0.05	0.01
α_{ARMA}	0.01	0.01
$\hat{a}_1$ (a_1=1)	0.8812	0.8858
$\hat{b}_0$ (b_0=1500)	1501	1501
performance	8.13e-04	3.08e-04

Table 4.9 EEM identification results of simulation 5



(a) (b)

Figure 4.10 Model validation after EEM identification of simulation 5
(a) System output h(k) when the input is [0, 3e-05]
(b) h(k) (solid) versus $\hat{h}$(k) (dashed)

OEM

$$\hat{h}(k) = \hat{a}_1 \hat{h}(k-1) + \hat{b}_1 \phi(k) + N_1[\hat{h}(k-1)] \tag{4.16}$$

The neural network structure is: $N_{1\,5\,5\,1}$
The learn parameters and the estimated ARMA parameters are gathered in table 4.10. The validation after the last run is shown in figure 4.11.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	0.001	0.001
α_{ARMA}	0.001	0.001
$\hat{a}_1$ (a_1=1)	0.8881	0.8881
b_0 (b_0=1500)	1490	1487
performance	1.77e-03	4.19e-04

Table 4.10 OEM identification results of simulation 5

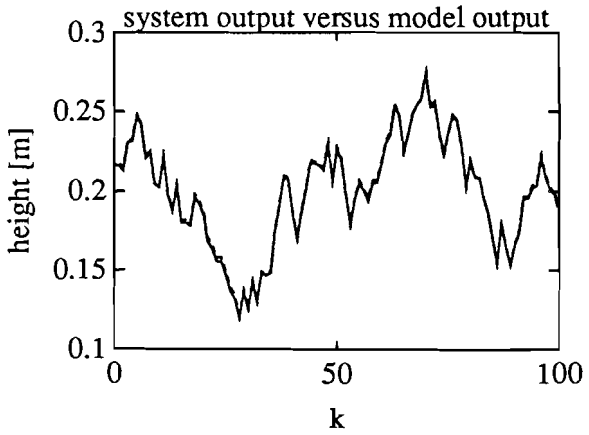


Figure 4.11 Model validation after OEM identification of simulation 5

As can be seen the model describes the system behaviour perfectly. It should be noticed that a linearisation is done automatically, so the a_1 parameter changes from 1 to 0.8881. The neural network model only models the nonlinear part.

4.2 Simulation results of systems with additive white noise

In figure 4.12 a system is given when its output is disturbed by random, uniformly distributed white noise. On account of time limitations, only the result of the first three simulation, disturbed by noise, are given here.

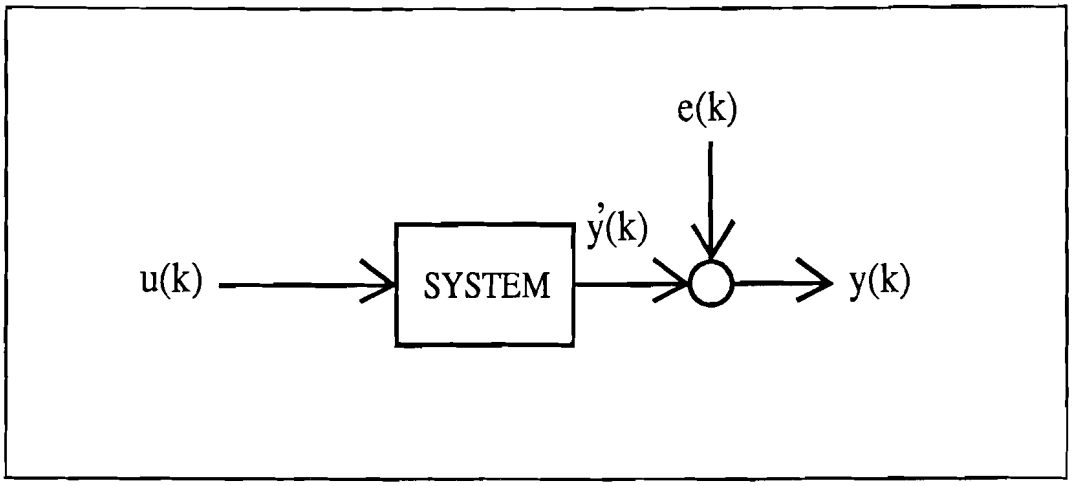


Figure 4.12 A system when its output is disturbed by additive white noise

Simulation 6

The system to be identified is described by the difference equation

$$y(k)=0.3y(k-1)+0.6y(k-2)+f[u(k)]+e(k) \tag{4.17}$$

with

$$f[u(k)]=u(k)^3+0.3u(k)^2-0.4u(k)$$

$e(k)$ random, uniformly distributed white noise [-0.1,0.1]

training data : random, uniformly distributed noise in the interval [-1.5, 1.5]
validation data :

$$u(k)=\begin{cases} \sin(2\pi k/250) & 0\le k\le 250 \\ 0.75\sin(2\pi k/250)+0.25\sin(2\pi k/25) & 250<k\le 500 \end{cases}$$

EEM

$$\hat{y}(k)=\hat{a}_1 y(k-1)+\hat{a}_2 y(k-2)+N_1[u(k)] \tag{4.18}$$

The neural network structure is: $N_{1\,5\,5\,1}$
The learn parameters and the estimated a-parameters are gathered in table 4.11.
The validation results after the last run are shown in figure 4.13.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	0.1	0.01
α_{ARMA}	0.01	0.01
$\hat{a}_1$ (a_1=0.3)	0.2992	0.3056
$\hat{a}_2$ (a_2=0.6)	0.5770	0.5934
performance	2.14e-02	2.75e-03

Table 4.11 EEM identification results of simulation 6

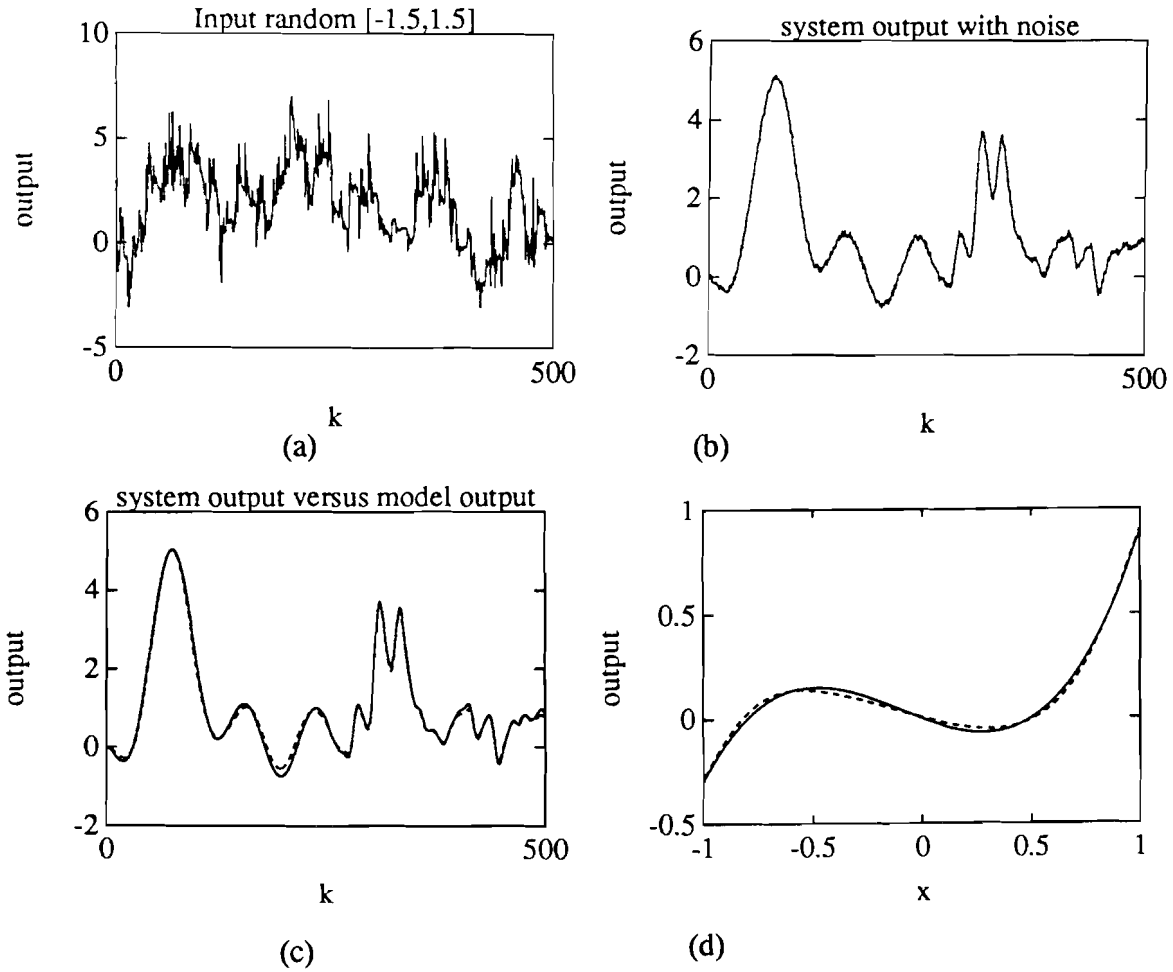


Figure 4.13 Model validation after EEM of simulation 6
(a) System output y(k) when the input is random [-1,1]
(b) System output during validation
(c) $\acute{y}$(k) (solid) versus $\hat{y}$(k) (dashed)
(d) f[u] (solid) versus N_1[u] (dashed)

OEM

$$\hat{y}(k)=\hat{a}_1\hat{y}(k-1)+\hat{a}_2\hat{y}(k-2)+N_1[u(k)] \tag{4.19}$$

The neural network structure is: $N_{1\,5\,5\,1}$

The learn parameters and the estimated A parameters are gathered in table 4.12.

	run 1
learn patterns	500
iterations	50
batch size	1
α_{NEURAL}	0.001
α_{ARMA}	0.001
$\hat{a}_1$ (a_1=0.3)	0.3033
$\hat{a}_2$ (a_2=0.6)	0.5960
performance	2.66e-03

Table 4.12 OEM identification results of simulation 6

The validation results after the last run are shown in figure 4.14.

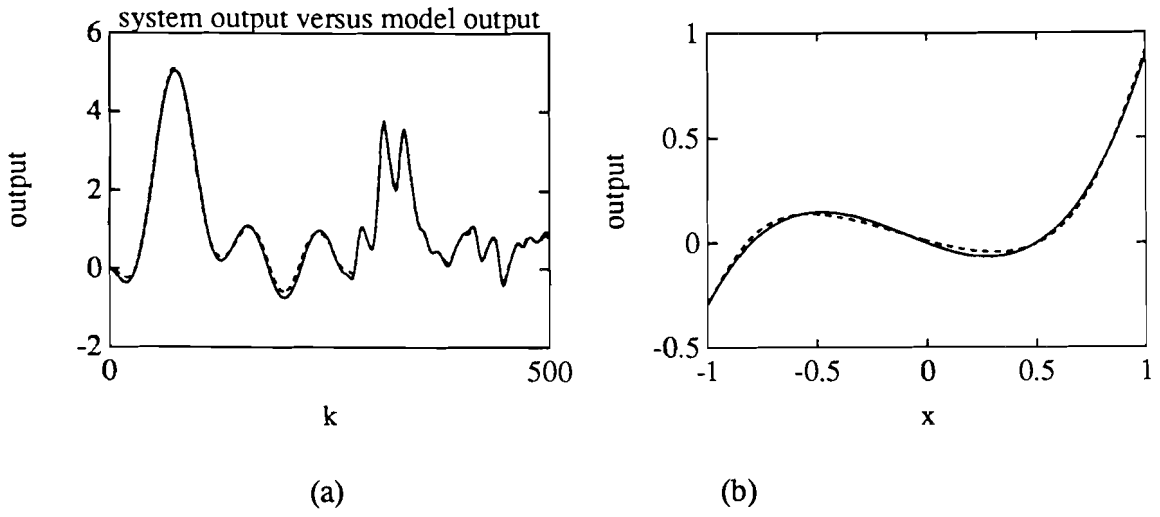


Figure 4.14 Model validation after OEM of simulation 1
(a) ý(k) (solid) versus ŷ(k) (dashed)
(b) f[u] (solid) versus N_1[u] (dashed)

The end performance is not as good as in the noise free case. An extension of the dataset, however, from 500 to 1000 samples leads to similar results as in the noise free case.

Simulation 7

The system to be identified is described by the difference equation

$$y(k)=f[y(k-1),y(k-2)]+u(k)+e(k) \tag{4.20}$$

with

$$f[y(k-1),y(k-2)]=\frac{y(k-1)y(k-2)[y(k-1)+2.5]}{1+y(k-1)^2+y(k-2)^2}$$

$e(k)$ random, uniformly distributed noise [-0.3,0.3]

training data : random, uniformly distributed noise in the interval [-2,2]
validation data :

$$u(k)=\sin(2\pi k/25)$$

EEM

$$\hat{y}(k)=N_1[y(k-1),y(k-2)]+\hat{b}_0 u(k) \tag{4.21}$$

The neural network structure is: $N_{2\,5\,5\,1}$
The learn parameters and the estimated b_0 parameter are gathered in table 4.13.

	run 1	run 2
learn patterns	1000	1000
iterations	25	25
batch size	1	1
α_{NEURAL}	0.05	0.01
α_{ARMA}	0.01	0.01
$\hat{b}_0$ (b_0=1)	1.013	1.011
performance	1.061	2.91e-02

Table 4.13 EEM identification results of simulation 7

The validation results after the last run are shown in figure 4.15.

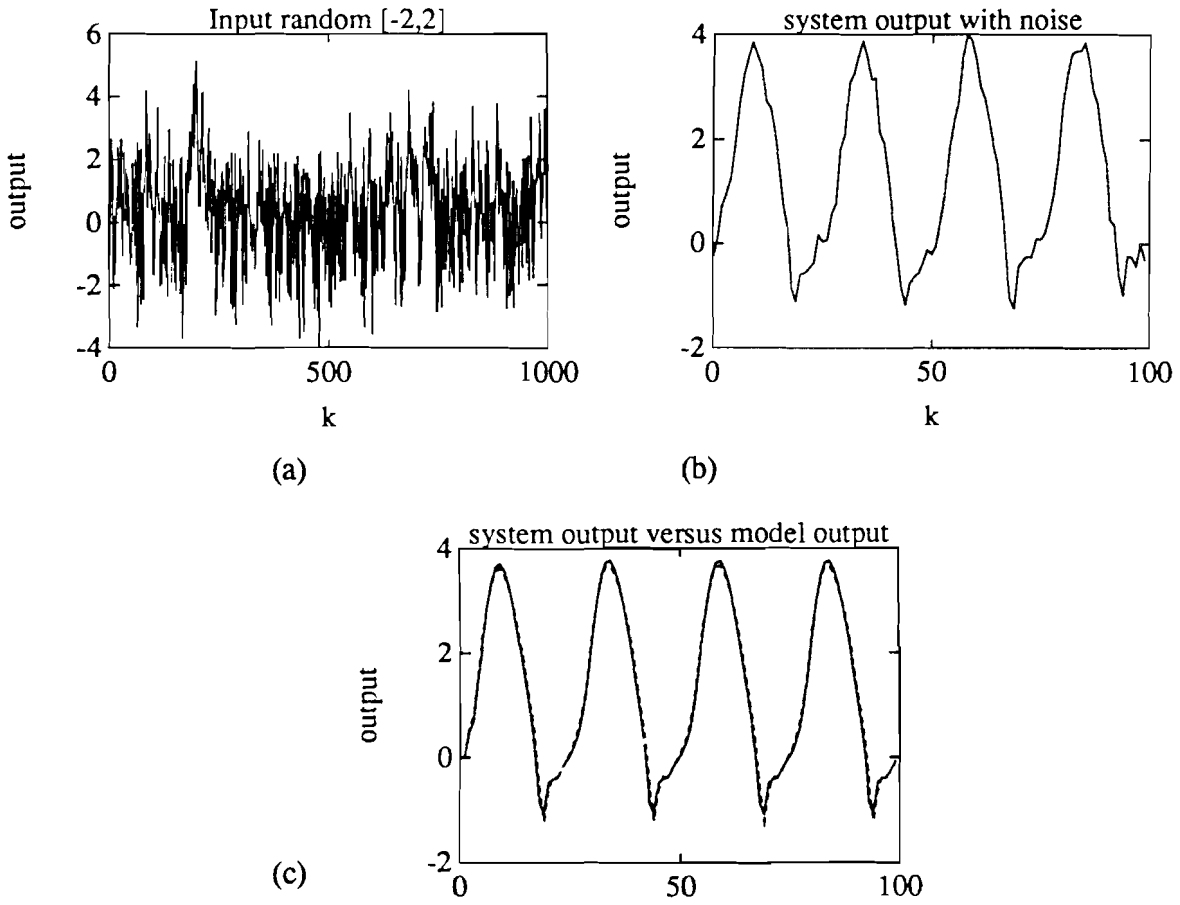


Figure 4.15 Model validation after EEM identification of simulation 7
(a) System output y(k) when the input is [-2,2]
(b) System output y(k) disturbed by noise
(c) Noise free output ŷ(k) (solid) versus model output y(k) (dashed)

OEM

$$\hat{y}(k)=N_1[\hat{y}(k-1),\hat{y}(k-2)]+\hat{b}_0 u(k) \tag{4.22}$$

Neural network structure $N_{2\,5\,5\,1}$

The learn parameters and the estimated b_0 parameter are gathered in table 4.14, and the simulation results after the last run are shown in figure 4.16.

	run 1	run 2
learn patterns	1000	1000
iterations	25	25
batch size	1	1
α_{NEURAL}	0.001	0.001
α_{ARMA}	0.001	0.001
$\hat{b}_0$ (b_0=1)	0.9968	0.9989
performance	3.7e-02	5.97e-03

Table 4.14 OEM identification results of simulation 7

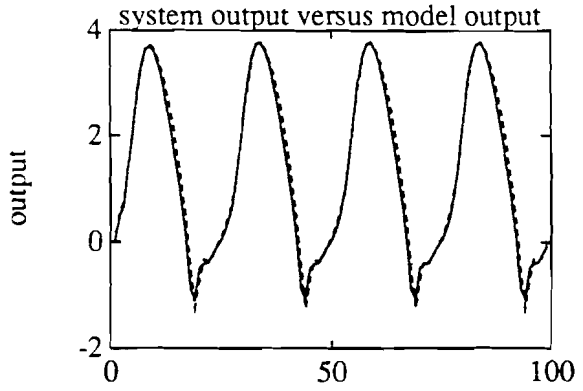


Figure 4.16 Model validation after OEM identification of simulation 7
ý(k) (solid) versus ŷ(k) (dashed)

As can be seen the identification of systems with noise added to the output is no problem due the fact that a OEM identification is used. The model indeed models the noise free system (ý(k)). A comparison with the noise free case (page 39) learns that one extra run is necessary to obtain the same (even lower) performance. Note that there is a substancial improvement in OEM compared to EEM due to proper noise modelling.

Simulation 8

In this simulation the system is a model 3 type of the form:

$$y(k) = f[y(k-1)] + g[u(k)] + e(k) \tag{4.23}$$

with

$$f[y(k-1)] = \frac{y(k-1)}{1+y(k-1)^2}$$

$$g[u(k)] = u(k)^3$$

$e(k)$ random, uniformly distributed noise in the interval [-0.5,0.5]

training data : random, uniformly distributed noise in the interval [-2, 2].
validation data :

$$u(k) = \sin(2\pi k/25) + \sin(2\pi k/10)$$

EEM

$$\hat{y}(k) = N_1[y(k-1)] + N_2[u(k)] \tag{4.24}$$

The neural network structures are: $N_1 = N_{1\,5\,1}$ and $N_2 = N_{1\,5\,1}$
The learn parameters are gathered in table 4.15.

	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	0.01	0.01
α_{ARMA}	--	--
performance	5.997	7.37e-03

Table 4.15 EEM identification results of simulation 8

The validation results after the last run are shown in figure 4.17.

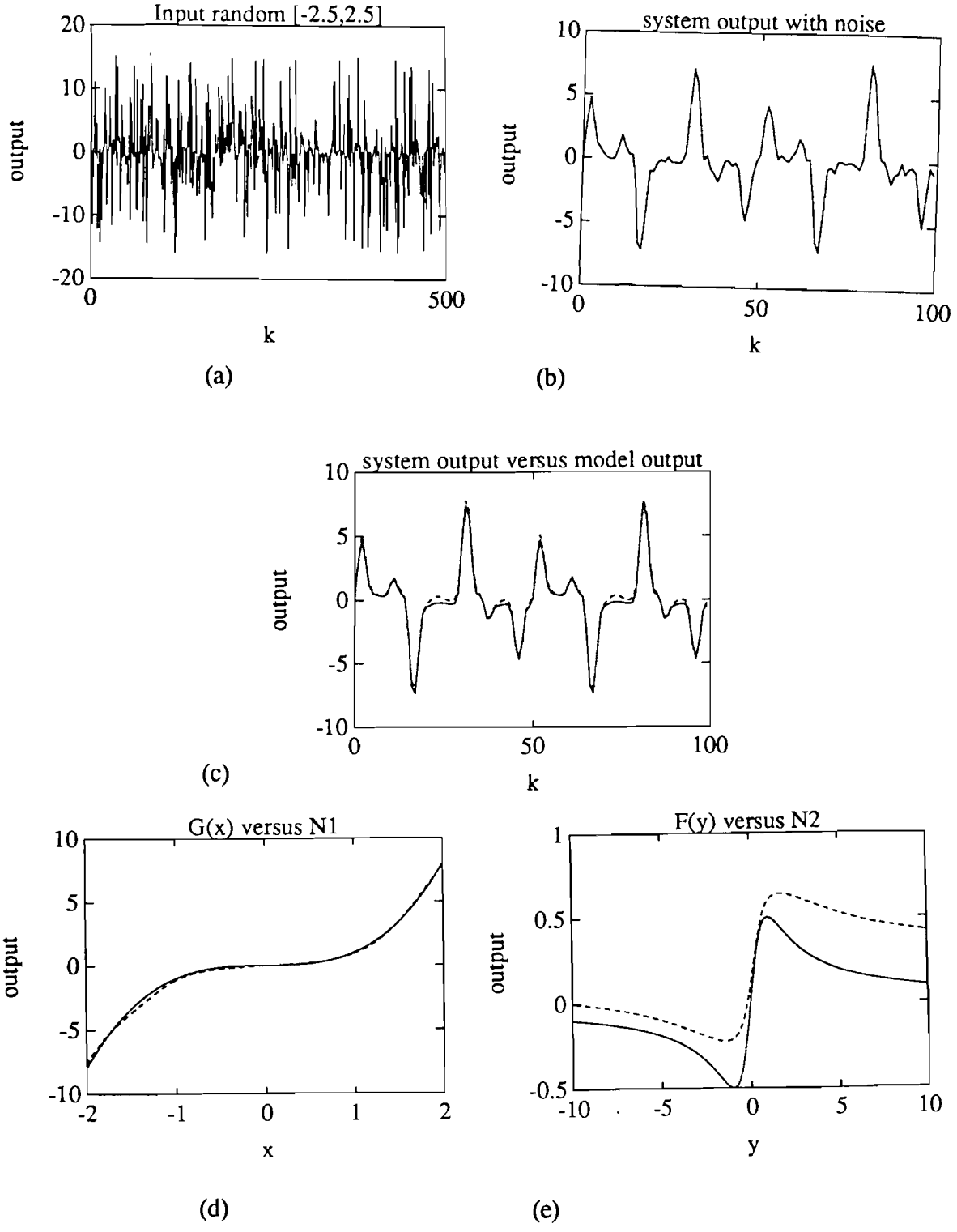


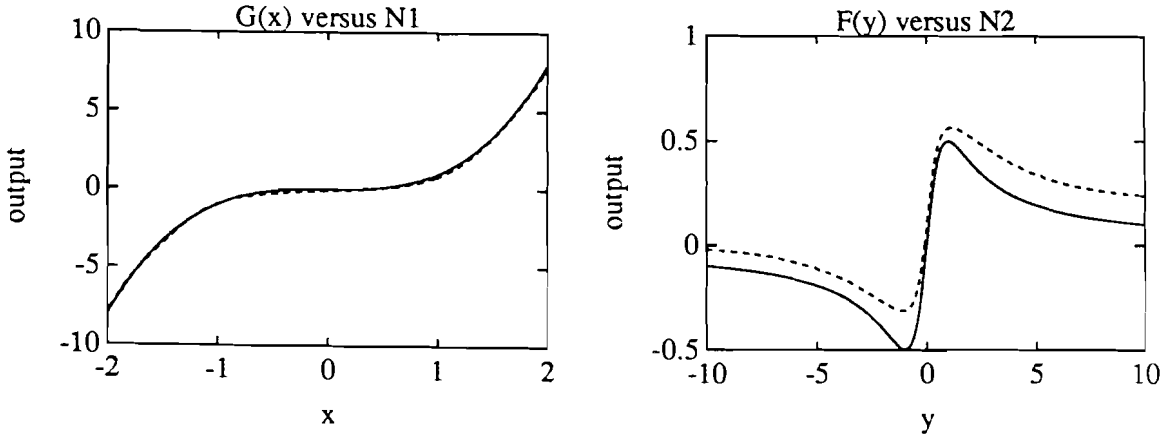
Figure 4.17 Model validation after EEM identification of simulation 8
(a) System output y(k) when the input is random [-2,2]
(b) System output during validation
(c) y(k) (solid) versus $\hat{y}(k)$ (dashed)
(d) g[u] (solid) versus N_2[u] (dashed)
(e) f[y] (solid) versus N_1[y] (dashed)

$$\hat{y}(k) = N_1[\hat{y}(k-1)] + N_2[u(k)] \tag{4.25}$$

The neural network structures are: $N_1 = N_{1\,5\,5\,1}$ and $N_2 = N_{1\,5\,5\,1}$
The learn parameters are gathered in table 4.16. The validation results after the last run are shown in figure 4.18.

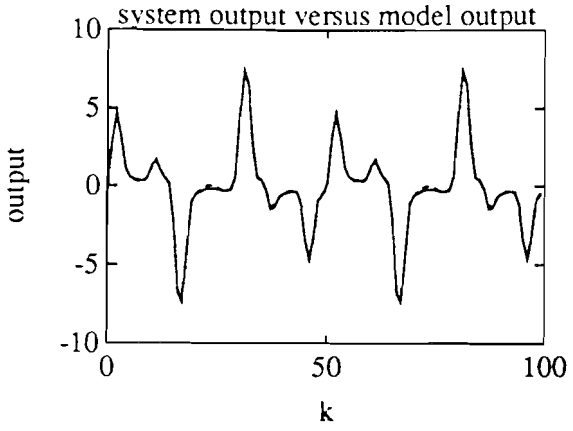
	run 1	run 2
learn patterns	500	500
iterations	50	50
batch size	1	1
α_{NEURAL}	1.0e-03	1.0e-03
α_{ARMA}	--	--
performance	3.19e-03	9.15e-05

Table 4.16 OEM identification results of simulation 8



(a) (b)

Figure 4.18 Model validation after OEM identification of simulation 8
(a) g[u] (solid) versus N_2[u] (dashed)
(b) f[y] (solid) versus N_1[y] (dashed)



(c)

Figure 4.18 Model validation after OEM identification of simulation 8
(c) y(k) (solid) versus $\hat{y}(k)$ (dashed)

4.3 Comparison with results obtained from the literature

In [12] the same simulations were done (only the noise free case). Here a similarly high performance is obtained. The way, however, the results are obtained is quite different.

- The neural network structures used here, contain considerable less parameters. Here neural network structures of $N_{i0\,5\,5\,1}$ are used and in the literature $N_{i0\,20\,10\,1}$ networks were used (i0 is the number of inputs to the neural network and depends on the simulation). This means about 200 parameters less.
- Also a smaller data set is used. Here 500 different samples are used, which are repeated 200 times to train the network. In the literature 100.000 different data samples were used. This makes the training of practical systems possible.
- When the simulation systems contain linear parameters, they are estimated with the nonlinear parameters simultaneously. Narendra and Parthasarathy did not estimate the linear part of the system, they set the linear parameters a priori to the real values. As can be seen in the simulations, the linear parameters are estimated very good.
- In contrast to the literature, also the OEM is used during training so a good simulation model will be obtained. Especially in the case when the system output is disturbed by noise, the OEM gives an improvement over the EEM. It

should be noticed that the special identification procedure (first the EEM and then the OEM) gives no improvement in the noise free case. The final performance is not better when the OEM is used in stead of another run with the EEM. In the case where the system output is disturbed by noise, however, the final performance obtained with the OEM is better than another run(s) with the EEM.

5 IDENTIFICATION OF THE WATER-VESSEL PROCESS

5.1 Description of the water-vessel process

The water-vessel process is a laboratory process, that is used by students during practical training to study some aspects of identification and control theories. The process consists of three vessels placed above each other. Water is pumped from a supply vessel into the top vessel and flows via the middle vessel into the lower vessel. From the lower vessel the water flows into the supply vessel. A scheme of the process has been drawn in figure 5.1.

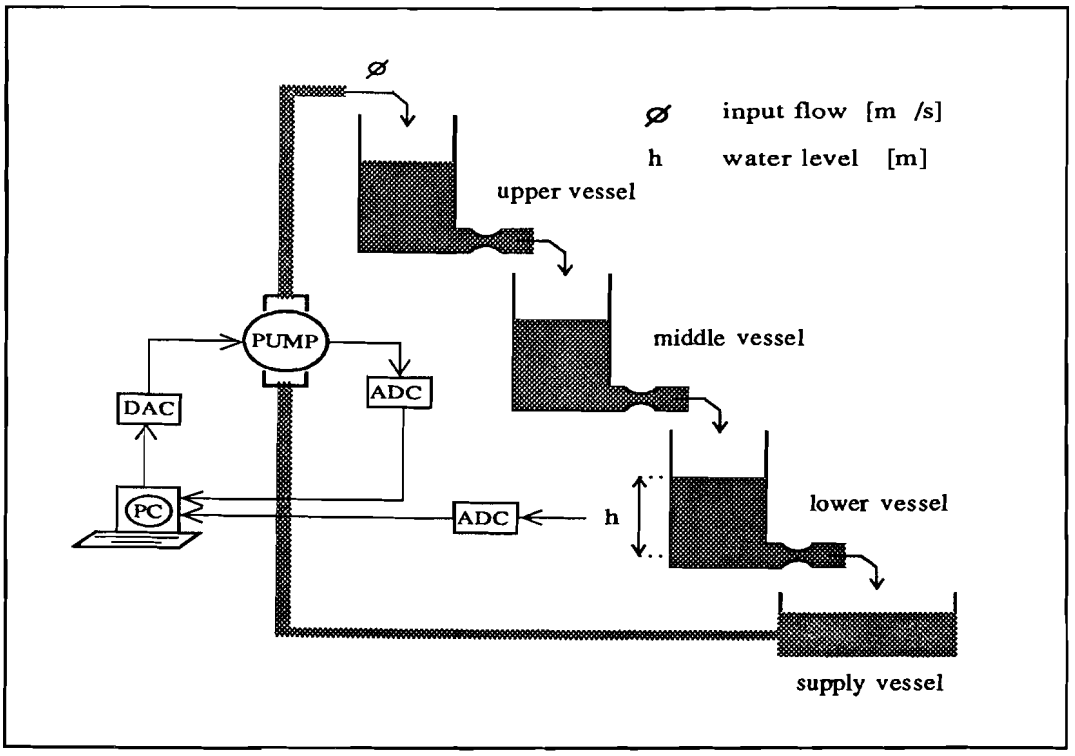


Figure 5.1 Scheme of the water-vessel process

In order to be able to measure and control the process, extra equipment has been added to the process.

- A level sensor for the lower vessel, to measure the water height.
- A rotation speed detector for the determination of the water flow.
- A personal computer (PC) to control the pump and store the process measurements.
- D/A and A/D converters for the interface between the PC and the process.

A more detailed description of the process and the peripherals is given by Liebregts [2].

The water level measurements are disturbed by:
- adhesion; for descending water the measured level is too high, which results in a hysteresis of maximal 2 [mm].
- the bubbles which appear when the water drops into the vessel.

The relation between the ADC value and the actual water height is:

$$h = 1.7\,10^{-4} ADC_h \tag{5.1}$$

where ADC_h discrete output value of the ADC
h actual water height in the lower vessel [m]

The water is pumped into the upper vessel by a roller pump. The flow is not constant but fluctuates around an average level. These fluctuations are very fast compared to the dynamics of the water vessels, so they can be neglected. The input voltage of the pump is, however, not a good measure for the average flow. The rotation speed of the pump is a much better measure for the average flow, so it will be measured and used as the model input. The measurement of the rotation speed is six times oversampled so a good value for the average flow can be calculated by averaging over a sample interval. The relation between the actual flow and the output of the ADC is:

$$\phi = 1.04\,10^{-8} ADC_\phi - 5.6\,10^{-7} \quad (\pm 4\%) \tag{5.2}$$

where ADC_ϕ discrete output value of the ADC
ϕ actual flow into the upper vessel [m^3/s]

Figure 5.2 gives a block diagram of the process with all the disturbances.

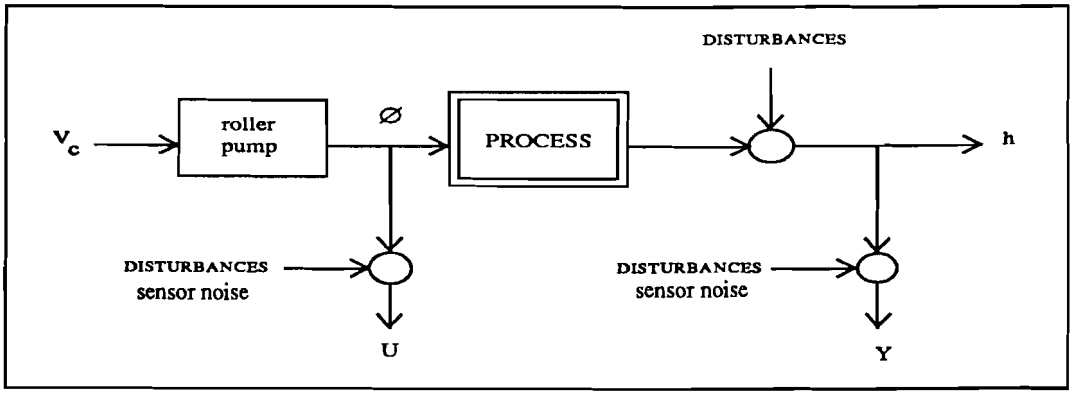


Figure 5.2 Block diagram of the water-vessel process

The input to the process is the roller pomp voltage and the output is the water height in the lower vessel, however, for the model the water flow will be the input. As can be seen in appendix B one vessel can theoretically be described by a first order nonlinear difference equation:

$$h(k) = a_1 h(k-1) + b_0 \phi(k-1) + f[h(k-1)] \tag{5.3}$$

where ϕ(k) flow into the vessel [m^3/s]
h(k) water level in the vessel [m]

While the water-vessel process contains three vessels, described by a first order nonlinear difference equation, the total process is a third order nonlinear process. However, the high frequency behaviour of the upper vessel (it has the smallest time constant) will be filtered by the middle and the lower vessel, and the sample frequency is too low to 'see' the dynamics of the upper vessel. Thus the water-vessel process can be described by a second order nonlinear difference equation.

5.2 Previous identification results

The previous identifications are all based upon linear identification techniques and can directly be used to initialize the ARMA parameters of the model. The results mentioned here are obtained by Lof [1]. Lof divides the process working range in 39 equal intervals and investigates how the ARMA parameters of the process change when different working points are passed through. The control voltage V_c passes through the entire working range from 0.6 to 3.9 [V] (ϕ is about 0.5 to 3.5 [*10e-5 m^3/s]). The model used is given by the subjoined difference equation.

$$y(k) = a_1 y(k-1) + a_2 y(k-2) + b_0 u(k) + b_1 u(k-1) + b_2 u(k-2) \tag{5.4}$$

where u(k) water flow into the upper vessel [m^3/s]
y(k) water level in lower vessel [m]

For the exact lapse of the ARMA parameters will be referred to [1], only the upper and lower values are given here (table 5.1).

	b_0	b_1	b_2	a_1	a_2
minimum	-10	-110	370	1.43	-0.77
maximum	40	290	530	1.75	-0.53

Table 5.1 Results of an earlier identification

5.3 Identification of the water-vessel process

In contrast with linear identifications the entire working range will be modelled by a nonlinear model. As discussed Lof modelled the water-vessel process by a number of linear models, belonging to different working points. For this purpose a PRBNS signal had to be superpositioned onto a varying working point. As working point variation a sine wave has been chosen, because it could easily be removed by data filtering. Although this data is not suited for neural network training, it was used for identification. New experiments were not possible while the water-vessel process was broken down and there was no time to reconstruct it. Information about the experiment done by Lof:

- PRBNS period time : 15 [s]
- oversample rate : 6
- total measure time : 22:30 [h:m]
- sine wave range : [0.9 , 2.6] [V]
  [0.86, 3.14] *10e-5 [m^3/s]
- PRBNS range : [-0.3 , 0.3] [V]
  [-0.36, 0.36] *10e-5 [m^3/s]

The data, as this is direct descended from the A/D converters, is plotted in figure 5.3.

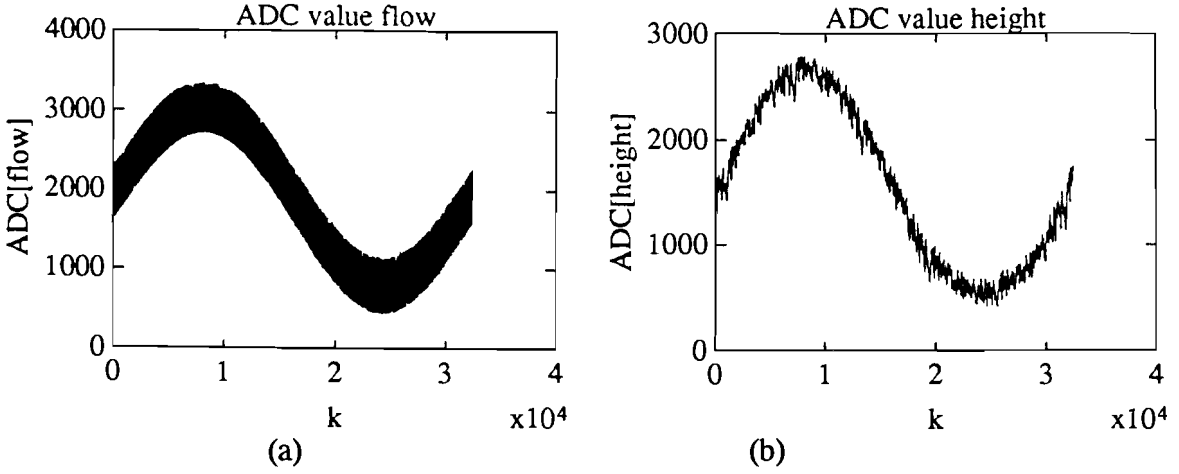


Figure 5.3 The raw data
(a) data from the rotation sensor
(b) data from the level sensor

First the data can be converted to the real water flow and water height with formula 5.1 and 5.2 (figure 5.4).

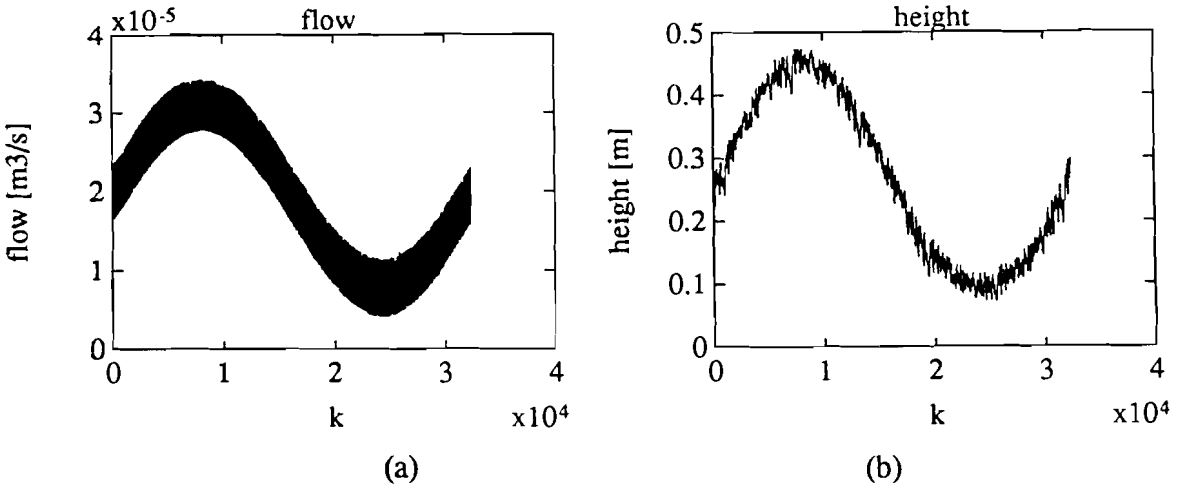


Figure 5.4 The actual data
(a) water flow ϕ [m³/s]
(b) water height h [m]

To make the data useful for nonlinear system identification with neural networks, data processing is necessary.

- Peak shaving is not necessary because the data contains no peaks.
- The delay time is about 7.5 [s], so the data must be corrected for this delay time.
- The oversampled rotation speed (flow) measurements have to be averaged.
- The input and output values have to be scaled to increase accuracy. The scaled input and output ranges are [-1,1].

The processed data is presented in figure 5.5.

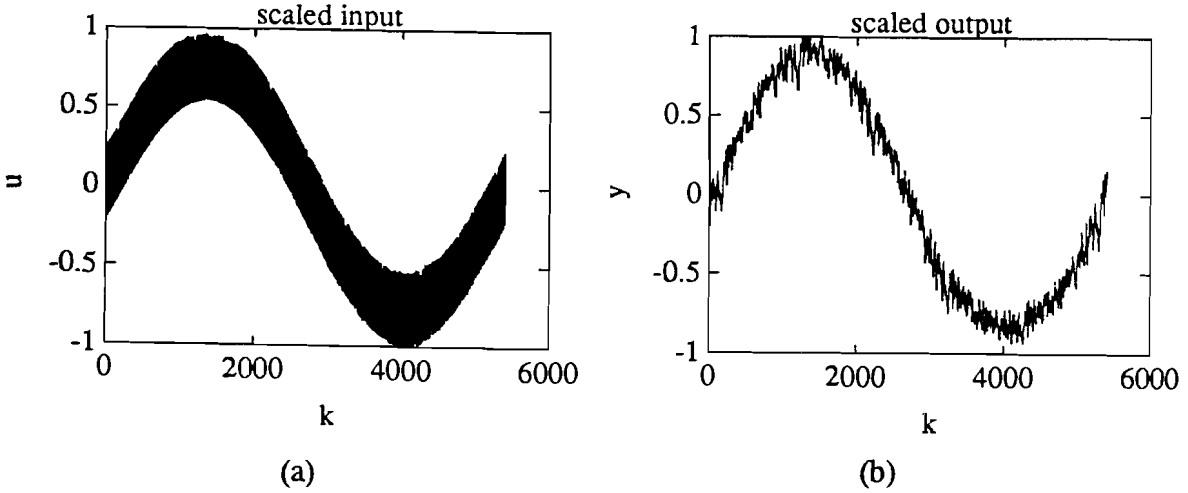


Figure 5.5 The scaled data
(a) input u
(b) output y

The relation between the scaled input and the actual flow is given by formula 5.5.

$$\phi(k) = 1.5335\,10^{-5}(u(k) + 1.2687) \tag{5.5}$$

where ϕ(k) actual flow [m^3/s]
u(k) scaled input used for identification

The relation between the scaled input and the actual flow is given by formula 5.6.

$$h(k) = 0.2070\,y(k) + 0.2651 \tag{5.6}$$

where h(k) actual height [m]
y(k) scaled model output used for identification

There are two kind of identifications done:
1. Without the use of a priori information.
2. With a prior knowledge about the auto regressive parameters from earlier identifications.

1. This identification of the water-vessel process makes no use of a priori knowledge. The identification is divided in two phases. During the first identification phase the ARMA parameters are estimated. This estimation is very fast comparing to the estimation of the neural network parameters. The results obtained are:

   $a_1 = 1.474$ $b_0 = 35$
   $a_2 = -0.5111$ $b_1 = 245$
   $b_2 = 377$

   As can be seen in table 5.1 most estimated ARMA parameters (except a_2) are within the variation range. In the second phase the ARMA parameters are initially set to the values obtained in the first phase.

   EEM

   As can be seen from equation 5.3, the nonlinear part is only a function of the past outputs.

$$\hat{y}(k) = \hat{a}_1 y(k-1) + \hat{a}_2 y(k-2) + \hat{b}_0 u(k) + \hat{b}_1 u(k-1) + \hat{b}_2 u(k-2) + N_1[y(k-1), y(k-2)] \tag{5.7}$$

The neural network structure is: $N_{2\,5\,5\,1}$

The learn parameters and the estimated ARMA parameters are gathered in table 5.2.

	run 1	run 2	run 3
learn patterns	5400	5400	5400
iterations	10	10	10
batch size	5400	5400	5400
α_{NEURAL}	0.2	0.2	0.2
α_{ARMA}	0.1	0.1	0.1
$\hat{a}_1$	1.471	1.471	1.741
$\hat{a}_2$	-0.5137	-0.5141	-0.5141
b_0	7.6	3.2	3.1
b_1	215	208	206
b_2	346	338	334
performance	4.41e-03	3.58e-04	2.97e-04

Table 5.2 EEM identification results of identification 1

While there was no validation data available, the data used for training was also used for validation. The validation results after run 3 are presented in figure 5.6.

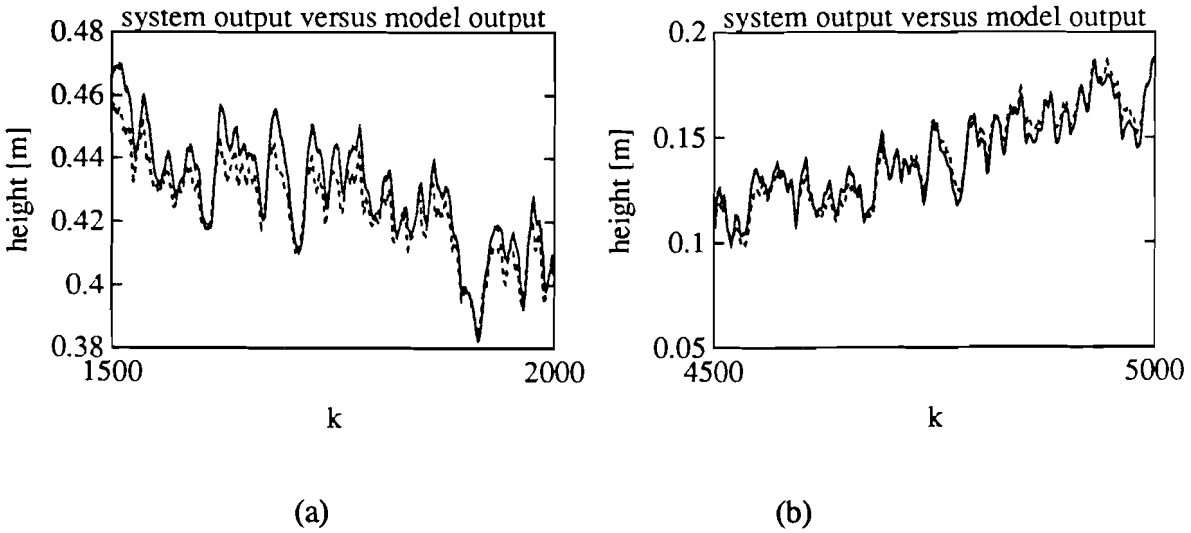


Figure 5.6 Model validation after EEM for identification 1
(a) y(k) (solid) versus $\hat{y}$(k) (dashed) for $1500 \leq k \leq 2000$
(b) y(k) (solid) versus $\hat{y}$(k) (dashed) for $4500 \leq k \leq 5000$

It should be noticed that the batch size is chosen to be 5400 (equal to the length of the data set). When the batch size was 1, there was no good fit over the entire operating range, while the data set is very extensive and the system passes very slow through the entire operating range. So the neural network model continually learns small pieces of the entire range, which results in a model belonging to the last part of the data set. To overcome this problem, the calculation of the gradient is only once over the entire data set, so the batch size is 5400.

OEM

$$\hat{y}(k) = \hat{a}_1\hat{y}(k-1) + \hat{a}_2\hat{y}(k-2) + \hat{b}_0u(k) + \hat{b}_1u(k-1) + \hat{b}_2u(k-2) + N_1[\hat{y}(k-1), \hat{y}(k-2)] \tag{5.8}$$

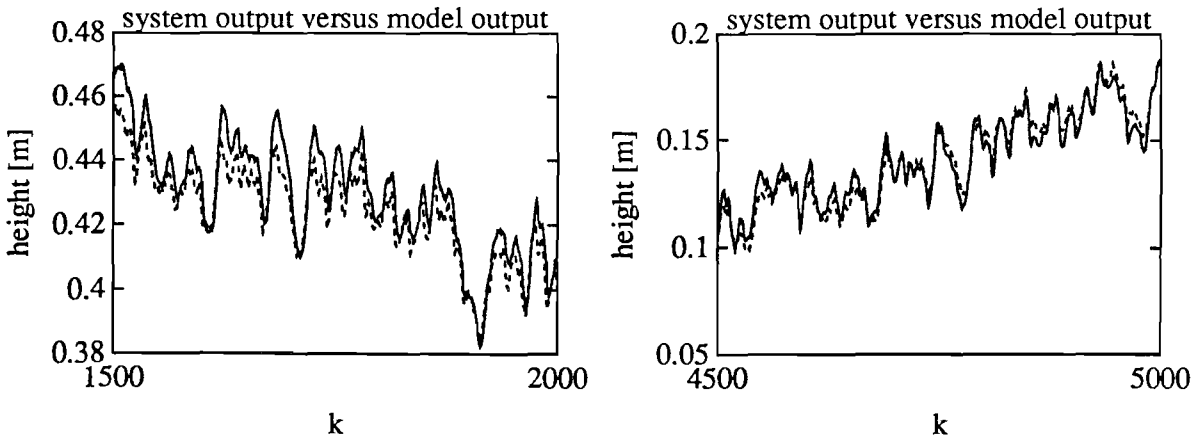
The neural network structure is: $N_{2\,5\,5\,1}$

The learn parameters and the estimated ARMA parameters are gathered in table 5.3.

	run 1	run 2
learn patterns	5400	5400
iterations	10	10
batch size	5400	5400
α_{NEURAL}	1.0e-03	1.0e-03
α_{ARMA}	1.0e-03	1.0e-03
$\hat{a}_1$	1.471	1.471
$\hat{a}_2$	-0.5141	-0.5141
b_0	3.2	3.2
b_1	206	206
b_2	334	334
performance	2.97e-04	2.97e-04

Table 5.3 OEM identification results of identification 1

The validation results after the OEM identification are shown in figure 5.7.



(a) (b)

Figure 5.7 Model validation after OEM for identification 1
(a) y(k) (solid) versus $\hat{y}$(k) (dashed) for $1500 \leq k \leq 2000$
(b) y(k) (solid) versus $\hat{y}$(k) (dashed) for $4500 \leq k \leq 5000$

2. Since the ARMA parameters are not constant over the entire work range, the ARMA parameters are initially set to the mean values of the estimated ARMA parameters. according to table 5.1 $\bar{a}_1 = 1.59$ $b_0 = 15$
$\bar{a}_2 = -0.65$ $b_1 = 90$
$b_2 = 450$

The identification procedure is divided in two parts, an EEM part and an OEM part.

EEM

$$\hat{y}(k) = \hat{a}_1 y(k-1) + \hat{a}_2 y(k-2) + \hat{b}_0 u(k) + \hat{b}_1 u(k-1) + \hat{b}_2 u(k-2) + N_1[y(k-1), y(k-2)] \tag{5.9}$$

The neural network structure is: $N_{2\,5\,5\,1}$
The learn parameters and the estimated ARMA parameters are gathered in table 5.4.

	run 1	run 2	run 3
learn patterns	5400	5400	5400
iterations	10	10	10
batch size	5400	5400	5400
α_{NEURAL}	0.2	0.2	0.2
α_{ARMA}	0.1	0.1	0.1
$\hat{a}_1$	1.593	1.594	1.594
$\hat{a}_2$	-0.6469	-0.6461	-0.6458
b_0	4.5	5.1	5.1
b_1	123	131	135
b_2	472	469	462
performance	2.90e-03	3.54e-04	2.97e-04

Table 5.4 EEM identification results of identification 2

The validation results after the EEM identification are shown in figure 5.8.

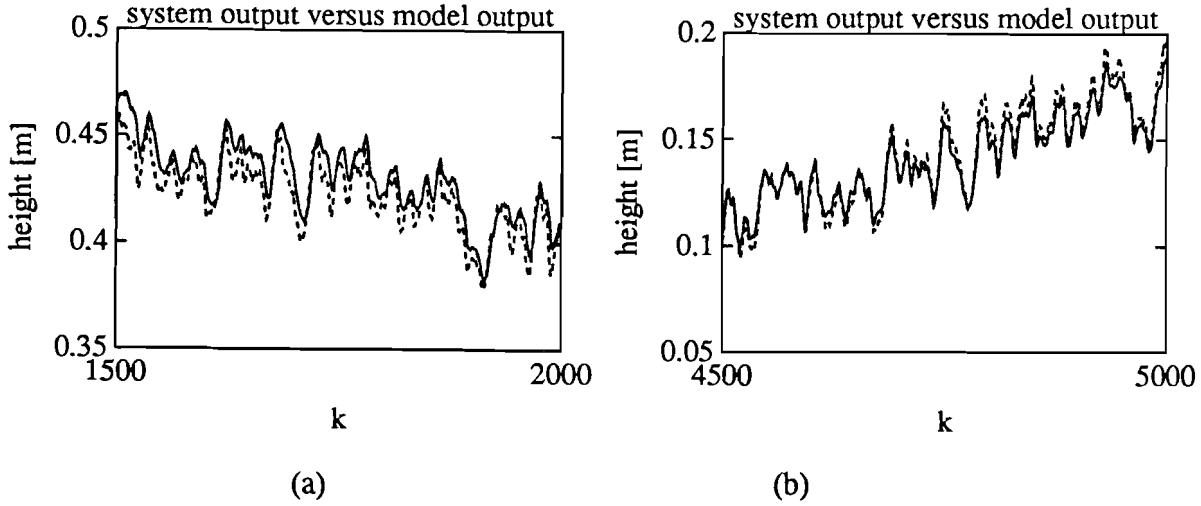


Figure 5.8 Model validation after EEM for identification 2
(a) y(k) (solid) versus $\hat{y}$(k) (dashed) for $1500 \leq k \leq 2000$
(b) y(k) (solid) versus $\hat{y}$(k) (dashed) for $4500 \leq k \leq 5000$

OEM

$$\hat{y}(k) = \hat{a}_1\hat{y}(k-1) + \hat{a}_2\hat{y}(k-2) + \hat{b}_0u(k) + \hat{b}_1u(k-1) + \hat{b}_2u(k-2) + N_1[\hat{y}(k-1), \hat{y}(k-2)] \quad (5.10)$$

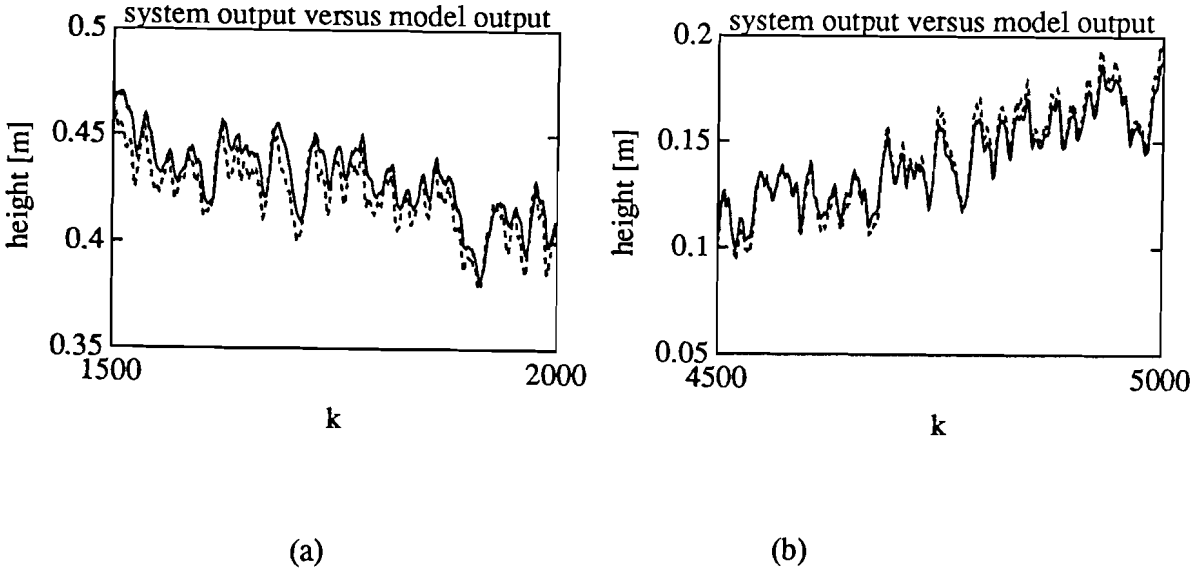
The neural network structure is: $N_{2\,5\,5\,1}$

The learn parameters and the estimated ARMA parameters are gathered in table 5.5.

	run 1	run 2
learn patterns	5400	5400
iterations	10	10
batch size	5400	5400
α_{NEURAL}	0.001	0.001
α_{ARMA}	0.001	0.001
a_1	1.594	1.594
a_2	-0.6457	-0.6457
b_0	5.2	5.2
b_1	135	135
b_2	462	462
performance	2.94e-04	2.94e-04

Table 5.5 OEM identification results of identification 2

The validation results after the OEM identification are shown in figure 5.9.



(a) (b)

Figure 5.9 Model validation after OEM for identification 2
(a) y(k) (solid) versus $\hat{y}(k)$ (dashed) for $1500 \leq k \leq 2000$
(b) y(k) (solid) versus $\hat{y}(k)$ (dashed) for $4500 \leq k \leq 5000$

5.3 Comparison with a linear identification

In this section the results obtained are compared to the results obtained by Lof [1]. The results are compared with a mean linear model (figure 5.10). The linear model is:

$$y(k) = a_1 y(k-1) + a_2 y(k-2) + b_0 u(k) + b_1 u(k-1) + b_2 u(k-2) \tag{5.11}$$

with $a_1 = 1.59$ $b_0 = 15$
$a_2 = -0.65$ $b_1 = 90$
$b_2 = 450$

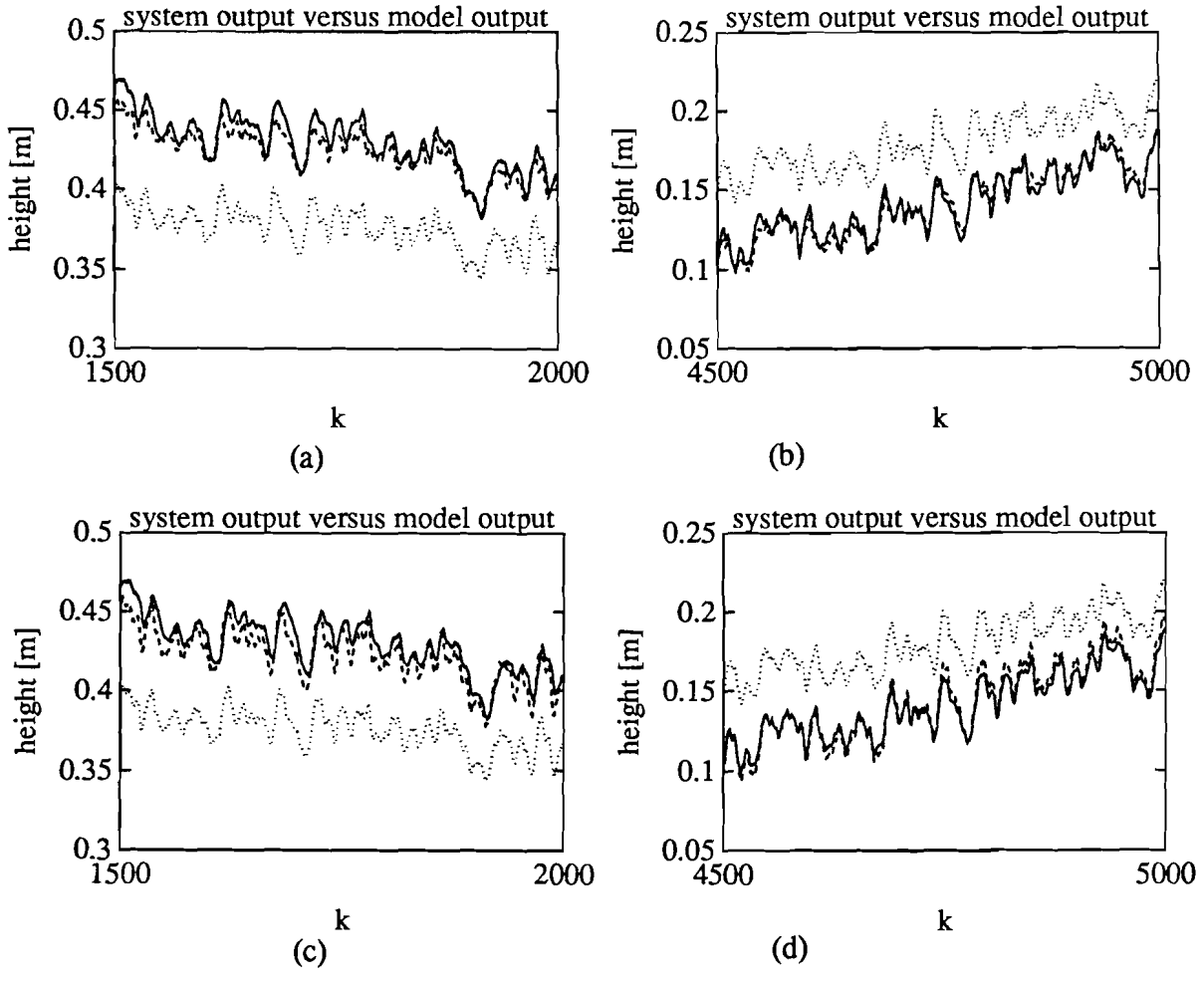


Figure 5.10 Simulation 1
(a) output system (solid), nonlinear model (dashed) and linear model (dotted) for $1500 \leq k \leq 2000$
(b) output system (solid), nonlinear model (dashed) and linear model (dotted) for $4500 \leq k \leq 5000$
Simulation 2
(c) output system (solid), nonlinear model (dashed) and linear model (dotted) for $1500 \leq k \leq 2000$
(d) output system (solid), nonlinear model (dashed) and linear model (dotted) for $4500 \leq k \leq 5000$

Conclusions

- The nonlinear model performs better over the entire range than the mean linear model. Even in the operating range for which the linear model holds, the nonlinear model gives the same performance, so the nonlinear model can replace 39 local linear models (used by Lof).
- In both identifications, with and without the use of a priori knowledge, a good model is obtained. When there is no a priori information available, it is better to estimate a linear model first (fast estimation compared to the nonlinear estimation) and then use this model to initialize a second nonlinear identification.
- By using a priori knowledge about the linear part of the system, the final model is found in only 50 steps (parameter modifications).
- Practice showed that the use of other neural network structures (with more parameters) give no improvement.
- Better results will be obtained when another data set is used, designed for neural network system identification (section 3.4).

6 CONCLUSIONS AND RECOMMENDATIONS

- It can be concluded that nonlinear dynamic system identification using neural networks leads to good models. These models can be obtained using a simple learning algorithm.

- The results obtained from simulations are in accordance to the results presented in the literature. However,
  - the neural network models used, contain considerable less parameters. $N_{i0\ 5\ 5\ 1}$ networks instead of $N_{i0\ 20\ 10\ 1}$ networks are used (i0 is the number of inputs).
  - a smaller data set is used, 500 samples in stead of 100.000 samples. This facilitates the identification of real systems.
  - the linear as well as the nonlinear parameters of the system model are estimated simultaneously.
  - the output error model is used during identification to obtain a good simulation model of the system. Especially in the case when the system output is disturbed by noise, the OEM gives an improvement over the EEM.

- As can be seen from the identification of the water-vessel process, it is advantageous to use a nonlinear identification technique using neural network models. Only one model can describe the process behaviour over the entire operating range in stead of 39 linear models. It is expected that even better results will be obtained when the input signal (during training) is a random, uniformly distributed signal over the entire working range.

- The use of a universal model which contains both a linear and a nonlinear model has proven to be valuable since the final model of the water-vessel process was found in only a few steps (about 50 parameter modifications).

- The application of the output error identification model with the neglect (section 3.2) leads to good results when the learnvelocity factors α_{NEURAL} and α_{ARMA} are chosen to be low (factor 10 to 100 with regard to the equation error model). To overcome the special identification procedure in the future, it is advisable to write a special algorithm for the recurrent multilayer neural network. Especially in the case when Newton-like algorithms will be applied, the total learning time will be acceptable.

- For the extension of the algorithm from SISO to MIMO, only the linear part of the learning algorithm has to be modified since the nonlinear part (neural network) is already MIMO.

REFERENCES

[1] Lof, L.D.E.
Parametric uncertainty modelling and identification for μ based control systems
Department of Electrical Engineering, Measurement and Control section,
Eindhoven University of Technology, December 1991

[2] Liebregts, W.R.H.M.
A new identification technique for H_∞ robust design: identification of the water-vessel process
Department of Electrical Engineering, Measurement and Control section,
Eindhoven University of Technology, October 1991

[3] Willis, M.J. et al
Artificial neural networks in process engineering
IEEE proceedings-D vol.138 no.3 May 1991 pp256-266

[4] Giacomin, J.
Neural network simulation of an automotive shock absorber
Engng Applic. of Artif. Intell. vol.4 no.1 1991 pp59-64

[5] Narendra, K.S., Parthasarathy, K.
Gradient methods for the optimization of dynamical systems containing neural networks
IEEE Trans. on Neural Networks vol.2 no.2 March 1991 pp252-262

[6] Yuan Zhen-Dong
Insight into neural network models for nonlinear system identification
Internal report Linköping University, Sweden Januari 1991

[7] Pham, D.T., Liu, X.
State-space identification of dynamic systems using neural networks
Engng Applic. of Artif. Intell. vol.3 Sept 1990 pp198-203

[8] Chen, S., Billings, S.A., Grant, P.M.
Non-linear system identification using neural networks
Int. Journal of Control vol.51 no.6 1990 pp1191-1214

[9] Heasloop, D., Holt, B.R.
Neural networks for process identification
Int. Joint Conference on Neural Networks vol.3 1990 pp429-435

[10] Tollenaere, T.
SuperSAB:Fast adaptive Backpropagation with good scaling properties
Neural Networks vol.3 1990 pp561-573

[11] Bhat, N.V., Minderman, P.A., Mc Avoy, T., Wang, N.S.
Modelling chemical process systems via neural computation
IEEE Control Systems vol.10 no.3 April 1990 pp24-30

[12] Narendra, K.S., Parthasarathy, K.
Identification and control of dynamical systems using neural networks
IEEE Trans. on Neural Networks vol.2 no.2 March 1991 pp252-262

[13] Hecht-Nielsen, R.
Neurocomputing
Amsterdam, Addison-Wesley 1990, ISBN 0-201-09355-3

[14] Simpson, P.K.
Artificial neural systems
Frankfurt, Pergamon press 1990, ISBN 0-08-037894-3

[15] Narendra, K.S., Parthasarathy, K.
Neural networks in dynamical systems
SPIE vol.1196 1989 pp230-241

[16] Funahashi, K.
On the approximate realization of continuous mapping by neural networks
Neural networks vol.2 1989 pp183-192

[17] Chu, S.R., Shoureshi, R., Tenorio, M.
Neural networks for system identification
American Control Conference vol.1 1989 pp916-921

[18] Hakim, N.Z., Kaufman, J.J., Siffert, R.S., Meadows, H.E.
A neural network model for prediction error identification
Conference on Signals Systems and Components vol.1 1989 pp119-122

[19] Kung, S.Y., Hwang, J.N., Sun, S.W.
Efficient modelling for multilayer feed-forward neural nets
Conf. on Acoustics, Speech and Signals vol.4 1988 pp2160-2163

[20] Rumelhart, D.E., McClelland, J.L.
Parallel Distributed Processing vol.1
Cambridge MA:MIT press 1986

APPENDIX A Diversion of the backpropagation algorithm

The performance function is defined as:

$$J=\frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}\frac{1}{2}[\hat{y}_p(k)-y_p(k)]^2 \tag{A.1}$$

with
- $\hat{y}_p(k)$ model output
- $y_p(k)$ system output
- N batch size
- i_L number of outputs

The weights are modified according to the steepest descent method.

$$W^{new}=W^{old}-\alpha\nabla_W J \tag{A.2}$$

with
- W^{new} new weight parameters
- W^{old} old weight parameters
- α learnvelocity factor
- $\nabla_W J$ gradient of J w.r.t. W

Figure A.1 shows a part of a multilayer neural network, which is used to derive the backpropagation algorithm.

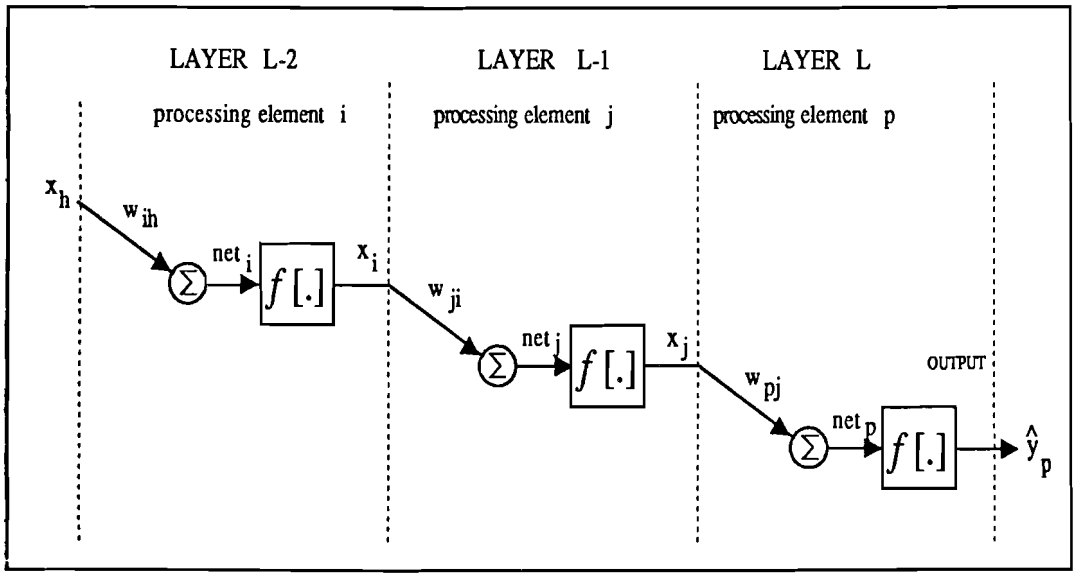


Figure A.1 Part of a multilayer neural network

$$\nabla_W J = \frac{\delta}{\delta W}\frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}\frac{1}{2}[\hat{y}_p(k)-y_p(k)]^2 \quad \Rightarrow \tag{A.3}$$

$$\frac{\delta J}{\delta W} = \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L} e_p(k)\frac{\delta \hat{y}_p(k)}{\delta W} \qquad \text{with} \quad e_p(k)=\hat{y}_p(k)-y_p(k) \tag{A.4}$$

1. $\mathbf{W}\in w_{pj}$ p: processing element of the output layer, layer L
   j: processing element of layer L-1

$$\begin{aligned}
\frac{\delta J}{\delta w_{pj}} &= \frac{1}{N}\sum_{k=1}^{N} e_p(k)\frac{\delta \hat{y}_p(k)}{\delta w_{pj}} \\
&= \frac{1}{N}\sum_{k=1}^{N} e_p(k)\frac{\delta \hat{y}_p(k)}{\delta net_p(k)}\frac{\delta net_p(k)}{\delta w_{pj}} \\
&= \frac{1}{N}\sum_{k=1}^{N} e_p(k) f'[net_p(k)]\, x_j(k) \\
&= \frac{1}{N}\sum_{k=1}^{N} \delta_p(k) x_j(k) \qquad \text{with} \quad \delta_p(k)=e_p(k)f'[net_p(k)]
\end{aligned}$$

where		
	$net_p(k)$	sum of all weighted inputs to processing element p
	f[.]	processing function of processing element p
	f'[.]	derivative of the processing function of processing element p

2. $\mathbf{W}\in w_{ji}$ j: processing element of layer (L-1)
   i: processing element of layer (L-2)

$$\begin{aligned}
\frac{\delta J}{\delta w_{ji}} &= \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L} e_p(k)\frac{\delta \hat{y}_p(k)}{\delta w_{ji}} \\
&= \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L} e_p(k)\frac{\hat{y}_p(k)}{x_j(k)}\frac{x_j(k)}{net_j(k)}\frac{net_j(k)}{\delta w_{ji}} \\
&= \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L} e_p(k) f'[net_p(k)]\, w_{pj} f'[net_j(k)]\, x_i(k) \\
&= \frac{1}{N}\sum_{k=1}^{N} x_i(k) f'[net_j(k)]\sum_{p=1}^{i_L}\delta_p(k) w_{pj} \\
&= \frac{1}{N}\sum_{k=1}^{N} \delta_j(k) x_i(k) \qquad \text{with} \quad \delta_j(k)=f'[net_j(k)]\sum_{p=1}^{i_L}\delta_p(k) w_{pj}
\end{aligned}$$

3. $W \in w_{ih}$ i: processing element of layer (L-2)
h: processing element of layer (L-3)

$$\begin{aligned}
\frac{\delta J}{\delta w_{ih}} &= \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L} e_p(k)\frac{\delta \hat{y}_p(k)}{\delta w_{ih}} \\
&= \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}\sum_{j=1}^{i_{L-1}} e_p(k)\frac{\hat{y}_p(k)}{x_j(k)}\frac{x_j(k)}{x_i(k)}\frac{x_i(k)}{net_i(k)}\frac{net_i(k)}{\delta w_{ih}} \\
&= \frac{1}{N}\sum_{k=1}^{N}\sum_{p=1}^{i_L}\sum_{j=1}^{i_{L-1}} e_p(k) f'[net_p(k)] w_{pj} f'[net_j(k)] w_{ji} f'[net_i(k)] x_h(k) \\
&= \frac{1}{N}\sum_{k=1}^{N} x_h(k) f'[net_i(k)] \sum_{j=1}^{i_{L-1}} w_{ji} f'[net_j(k)] \sum_{p=1}^{i_L} \delta_p(k) w_{pj} \\
&= \frac{1}{N}\sum_{k=1}^{N} x_h(k) f'[net_i(k)] \sum_{j=1}^{i_{L-1}} \delta_j(k) w_{ji} \\
&= \frac{1}{N}\sum_{k=1}^{N} x_h(k)\delta_i(k) \qquad \text{with} \quad \delta_i(k) = f'[net_i(k)] \sum_{j=1}^{i_{L-1}} \delta_j(k) w_{ji}
\end{aligned}$$

So in general:

$$\frac{\delta J}{w_{rq}} = \frac{1}{N}\sum_{k=1}^{N} \delta_r(k) x_q(k) \tag{A.5}$$

where q processing element of layer l-1
r processing element of layer l
x_q output of processing element q

When r is an processing element of a hidden layer:

$$\delta_r(k) = f'[net_r(k)] \sum_{s=1}^{i_{l+1}} \delta_s(k) w_{sr} \tag{A.6}$$

where s processing element of layer l+1

When r is an processing element of the output layer:

$$\delta_r(k) = [\hat{y}_r(k) - y_r(k)]\, f'[net_r(k)] \tag{A.7}$$

To clarify the backpropagation algorithm Narendra and Parthasarathy [12] give a blockscheme in which this backpropagation of the error is displayed.

APPENDIX B Model of a water-vessel

In figure B.1 is a schematic view of a water-vessel given.

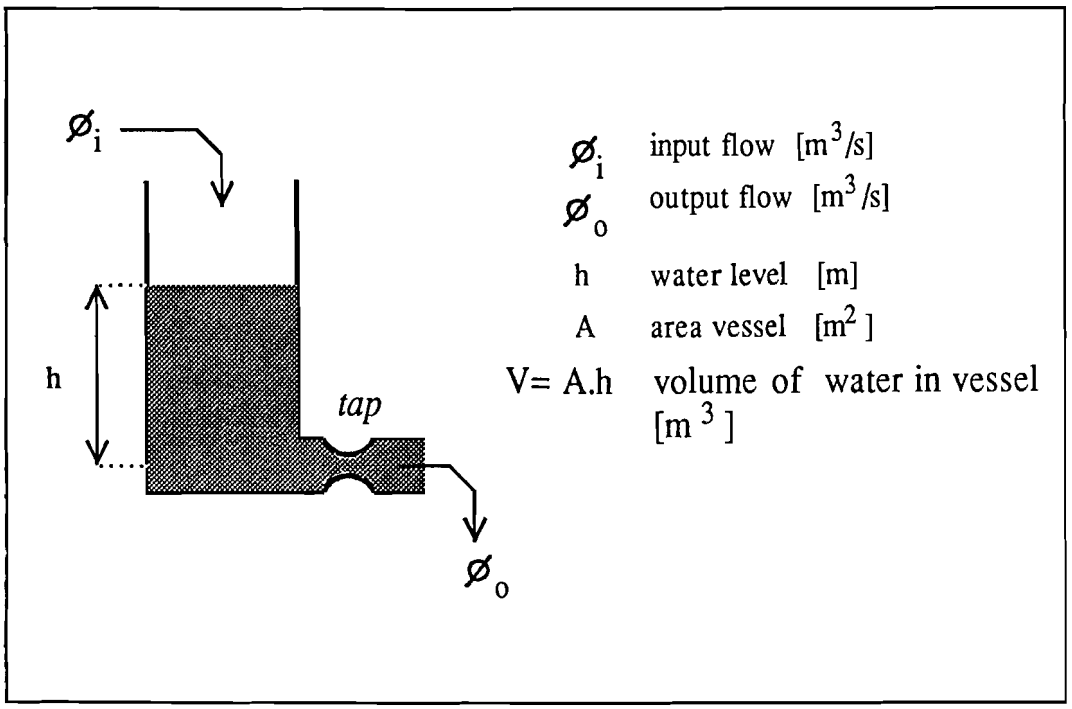


Figure B.1 Model of the watervessel

$$\phi_i - \phi_o = \frac{dV}{dt}$$

$$\phi_i - \phi_o = A\frac{dh}{dt} \qquad \text{with} \quad \phi_o = g[h]$$

The nonlinear relationship between Φ_o and h is given by Liebregts [2].

$$h = R_0\phi_o^2 + R_1\phi_o + D_0 \qquad \text{with} \quad \begin{aligned} R_0 &= 3.97e8 \\ R_1 &= 5.38e3 \\ D_0 &= 0.02 \end{aligned} \qquad \text{(B.1)}$$

With formula B.1 one can derive the subjoined equation.

$$\phi_o = \frac{-R_1 \pm \sqrt{R_1^2 - 4R_o(D_0 - h)}}{2R_0} \quad \wedge \quad \phi_o > 0$$

$$= \frac{-R_1}{2R_0}\left\{1 - \sqrt{(1 - 4D_0\frac{R_0}{R_1^2}) + 4\frac{R_0}{R_1^2}h}\ \right\}$$

$$= g\,[h]$$

So:

$$\phi_i(t) = A\frac{dh}{dt} + g\,[h]$$

Discretization:

$$\left.\frac{dh}{dt}\right|_{=kT} = \frac{h(kT+T) - h(kT)}{T}$$

The difference equation is:

$$\phi_i(k) = \frac{A}{T}\{h(k+1) - h(k)\} + g\,[h]$$

or:

$$h(k+1) = \frac{T}{A}\phi_i(k) + h(k) + f[h(k)]$$

So in general:

$$h(k) = a_1 h(k-1) + b_1 \phi_i(k-1) + f[h(k-1)]$$

$$a_1 = 1$$

$$b_1 = \frac{T}{A}$$

$$f[h(k-1)] = \frac{T}{A}\frac{R_1}{2R_0}\left[1 - \sqrt{(1 - 4D_0\frac{R_0}{R_1^2}) + 4\frac{R_0}{R_1^2}h(k-1)}\right] \qquad \text{(B.2)}$$