Neural networks in extremal seeking control

Joakim Kvistholm

Department of Automatic Control
Lund Institute of Technology
April 1999
Title and subtitle
Neural networks in extremal seeking control (Extremalsökning med regulatorer baserade på neuronalnät)

Abstract
The goal with this project is to examine how well controllers, that are based on neural networks, operates as controllers in extremal seeking control. The neural networks are based on the backpropagation algorithm and are implemented in Matlab. The controllers in this project are minimising the variance of the output from the process, so that the output lies as close as possible to a fixed reference value (which is a steady state minimum). The best controller is the controller that has the lowest summed variance (lowest losses). The performance of three neural controllers are compared with two other controllers: the constant extremum controller and the mixed extremum controller.
## Contents

1 Introduction .................................................. 1

2 Neural networks ................................................. 2
   2.1 Introduction to neural networks ....................... 2
   2.2 Transfer functions ..................................... 3
   2.3 Backpropagation ....................................... 5
   2.4 Bias ................................................... 9
   2.5 Momentum .............................................. 12

3 Extremal seeking control ...................................... 15
   3.1 The Hammerstein model ................................ 15
   3.2 Control objectives ................................... 16
   3.3 Admissible controls .................................... 17
   3.4 The constant extremum controller .................... 17
   3.5 The mixed extremum controller ....................... 18

4 Neural networks in Matlab .................................... 25
   4.1 The neural network .................................... 26
   4.2 Calculations of the neural network error, $E_i(k+1)$ 29
   4.3 Controller 1 based on neural networks ............... 42
   4.4 Controller 2 based on neural networks ............... 44
   4.5 Controller 3 based on neural networks ............... 45

5 Simulations on a process with known process parameters 47
   5.1 The constant extremum controller .................... 49
   5.2 The mixed extremum controller ....................... 51
   5.3 Controller 1 based on neural networks ............... 53
   5.4 Controller 2 based on neural networks ............... 61
   5.5 Controller 3 based on neural networks ............... 65
   5.6 Conclusions ........................................... 69

6 Summary and conclusions ..................................... 71

Bibliography ................................................... 72
1 Introduction

In many problems it’s assumed that a reference value is given. Controllers are often used to obtain a process output, that is as close as possible to a specified reference value. The process, that is studied, is given by a quadratic Hammerstein model, see section 3.1. Noise is introduced at the output of the dynamic process. In this project it’s desirable to obtain an output, that is as close as possible to the reference value, which is given by a steady state minimum. To track a varying extremum is called extremal seeking control. Controllers, that minimise the variance of the output around the reference value, are introduced, see section 3.2 and 3.5. These controllers are called minimum variance controllers. To make it possible to compare different controllers an optimisation criteria is formulated, see section 3.2. It’s based on a loss function. Two controllers in computer controlled systems, the constant extremum controller and the mixed extremum controller, are used in the simulations, see section 3.4-3.5 and section 5.1-5.2. The constant extremum controller has not taken the dynamic effects of the process into consideration, as the mixed extremum controller has done. This means that the mixed extremum controller has lower losses in the long run. The two controllers in computer controlled systems are then compared with three controllers, that are based on neural networks, see chapter 5.

The main goal of this project is to see how well neural networks operate as controllers in extremal seeking control. The theory behind neural networks and then especially the backpropagation algorithm is explained in chapter 2. The backpropagation algorithm is studied more closely, since it’s used in the controllers, that are based on neural networks. In chapter 4 a simpler example is used to test how well the neural network, can approximate the discontinuous function, that is given by the control law of the mixed extremum controller. Three controllers, that are based on neural networks, are implemented in Matlab, see chapter 4, and are later simulated in chapter 5. The results from the simulations are summarised in section 5.6.

The three controllers, that are based on neural networks: controller 1 (see section 4.3 and 5.3), controller 2 (see section 4.4 and 5.4) and controller 3 (see section 4.5 and 5.5), are working well. They have problems to follow the discontinuity in the control signal, see graph 5.3.2, graph 5.3.7 and formula (3.25). This fact increases the losses for the controllers, that are based on neural networks. Since a lesser part of the interval is effected, when a larger standard deviation of the noise, $\sigma_e$, is used, the controllers that are based on neural networks work better with a larger standard deviation in the process (3.3). The controllers, that are based on neural networks, don’t get enough training data at the end intervals, but this fact doesn’t effect the losses so much. Controller 1, controller 2 and controller 3 have losses that are somewhat larger than the mixed extremum controller, when $\sigma_e = 0.2$ and $\sigma_e = 1$, (with the exception of controller 3, when $\sigma_e = 1$) and have losses that are lower than the constant extremum controller, see table 5.6.1. Controller 1, when $\sigma_e = 1$, is the only controller, that is based on neural networks, with lower losses than the mixed extremum controller. Among the controllers that are based on neural networks, controller 2 has the best performance, when $\sigma_e = 0.2$. If only one of the controllers, that are based on neural networks, is used for both $\sigma_e = 0.2$ and $\sigma_e = 1$, controller 3 ought to be chosen, see section 5.6 and table 5.6.1.
2 Neural networks

In this chapter neural networks are described. The most commonly used neural networks are those which use backpropagation. They are used as a part of a controller in the simulations, which are described in a later chapter. That is why the backpropagation algorithm is so thoroughly explained. Questions like how many layers are necessary and how many neurons per layer are needed in the neural networks are touched upon. The material in this chapter is taken from Hertz, Krogh and Palmer (91) [1], Demuth and Beale (93) [4] and Bistedt (94) [5].

2.1 Introduction to neural networks

The ideas behind neural networks and then especially the neuron model comes from the human brain, see Hertz, Krogh and Palmer (91) [1] p. 1-10. A simplified model is made of the neurons and how they are connected to each other. The brain is an enormous system, where each neuron has a small task to perform. A lot of input signals are added to the neuron and if their sum is greater than a certain value, the neuron sends out a signal to another neuron. If this sum isn’t large enough, the input signals are absorbed by the neuron. The neural networks can be implemented with a computer and are used in control systems. Neural networks are good at finding patterns and to generalise, once they have been properly trained, see Demuth and Beale (93) [4] p. 1.1, 2.1, 2.16-2.17, 5.1 and 5.43. They can see structures in a large amount of data containing input signals and their corresponding output signals. They can also be used in non-linear systems, see Bistedt (94) [5] p. 9 and Demuth and Beale (93) [4] p. 2.16-2.17, 5.1 and 5.43. Since the neural networks can be non-linear and they can solve a non-linear problem at least to some extension, they are often used in non-linear systems. An example of a non-linear system, where a neural network can be used, is a text scanner, that identifies different letters in a text.

The disadvantages with the neural networks are that they often must be very large (with a lot of neurons in each layer) and that it takes a lot of time to train them properly. This requires fast computers or good integrated circuits. There is often a more direct method, which solves the problem better, but this solution is often extremely hard to find. In other words the neural network does not require as much knowledge of the problems, which are to be solved, as is necessary if the problems are solved in a more direct way.

There are no general rules of how the optimal neural network should be constructed for a general problem. It’s necessary to use the trial and error method in order to learn how to construct the optimal neural network, that solves a special problem. There are a few rules of thumb, that can be used in certain problems.

For instance, how many hidden layers are to be used in order to approximate a special class of functions $F; \{x_k\}$ to a given accuracy? The answer is that at most two hidden layers are necessary (two hidden layers and an output layer), if there are enough neurons in each layer. This is necessary if the function that is approximated changes from one function to another function at certain input values. If a continues function is approximated only one hidden layer (and of course the output layer) is needed. About the number of neurons, that are necessary in the different layers, it’s only possible to say that this number grows exponentially with the
number of inputs that are used in the neural network, see Hertz, Krogh and Palmer (91) [1] p. 142-144. A typical neuron can be seen in fig 2.1.1.

The conventional neural network often contains several layers, in which it’s possible to have any number of neurons. The neural networks consist of neurons and connections between the neurons in the different layers, see Bistedt (94) [5] p. 1-8. The neural networks can also contain bias, that are added to the neurons. The neurons have a transfer function that operates on the input signals and the bias. The transfer functions are in most cases either going to limit the outputs (from the neurons) or are linear functions. A neural network consists usually of one or two hidden layers and a output layer. A typical neural network is seen in fig 2.3.1.

![Neuron Diagram](image)

Fig 2.1.1. A typical neuron. It has the transfer function $g$ that operates on the incoming signals, $\xi_i$ and the bias $b_1$. The output $O_1$ is given by the expression, which can be seen inside the neuron. The input $\xi_i$ (for $i=1,2$ and 3) is multiplied with the weight $w_{1,i}$, before it becomes one of the inputs to the neuron.

2.2 Transfer functions

The transfer function in a neuron operates on the input signals to that neuron. Most transfer functions are non-linear. There are many transfer functions that can be used in a neural network, but if the backpropagation algorithm is used the function must be differentiable. The most commonly used functions are sigmoid functions, linear functions and different kind of step-like functions. Among the step-like functions the Heavyside’s function, which jumps from 0 to 1 at a specific input value, and the symmetric step function, which jumps from $-1$ to 1 at a specific input value, are worth mentioning. The sigmoid functions restricts the output signals to values between 0 and 1 or between $-1$ and 1. Two kind of sigmoid
function are of special interest, the logistic sigmoid function \( g(h) \) and tan-sigmoid function \( f(h) \), see Demuth and Beale (93) [4] p. 9.50 and 9.97. The formulas for these two functions are:

\[
g(h) = \frac{1}{1 + e^{-2\beta h}} \tag{2.1}
\]

\[
f(h) = \tanh(\beta h) \tag{2.2}
\]

The sigmoid functions get their name from their s-shape. The parameter \( \beta \) determines the maximal slope in the graph. These slopes increase with increasing \( \beta \), see fig 2.2.1.

Fig 2.2.1. The logistic sigmoid function can be seen to the left and tan-sigmoid function can be seen to the right. \( \beta \) is given the values 1.0, 0.5 and 0.25 in both graphs. As can be seen the sigmoid functions limit the output signals from the neurons.

The derivatives of the two sigmoid functions, given by the formulas (2.1) and (2.2), can be expressed by the function itself. This makes these functions very attractive to use, since the derivatives of the transfer functions are needed in the backpropagation algorithm. The derivatives are given by:

\[
g'(h) = 2\beta g(h)[1 - g(h)] \tag{2.3}
\]

\[
f'(h) = 2\beta [1 - f^2(h)] \tag{2.4}
\]
2.3 Backpropagation

Backpropagation is one of the most used algorithms in neural networks, see fig 2.3.1. The reasons are, for instance, that it's easy to implement and that the algorithm is simple. The algorithm is often based on a least mean square method. The material in this section and section 2.4 is mainly taken from Bistedt (94) [5] p. 1-8 and Hertz, Krogh and Palmer (91) [1] p. 115-148.

The main ideas behind backpropagation are:

1. A cost function (error function), $E$, is introduced with the goal to minimise the errors from the neural network with respect to the weights $w_{i,j}$ and $W_{i,j}$. The errors are related to the difference between the correct output and the output that is received. When this difference is minimised the cost function is also minimised.

2. The error function $E$ decreases most rapidly in the opposite direction of the gradient. It's important that a too large step isn't used, when the steepest descent (gradient descent) method is used. The gradient is therefore scaled with the value $\eta$. Intuitively it's evident that the gradient is a local phenomena and therefore it can only be used locally (in other words small steps are used). The gradient descent method can only be used on an error function $E$ that is differentiable. This restricts the transfer functions (in the different layers) to differentiable functions.

3. Since the output values are used first, it's necessary to first change the weights in the output layer, then in the hidden layers and finally in the 'input layer' (the first hidden layer). This step gives the algorithm it's name, backpropagation. Finally, the steps above are repeated until an error from the cost function $E$, that reaches an agreed level, is received.

The following analysis gives a more detailed explanation of how the algorithm works. First of all the different input patterns are named with different $\mu$'s. The superscript $\mu$ is connected with a special kind of input pattern, where $\mu$ can have the values 1, 2, 3 and so on. For instance, an input on the $k$:th place, when using the pattern $\mu$ is named $\xi_k^{(\mu)}$. Only a special case is studied, in which only two layers (a hidden layer and a output layer) is used, see fig 2.3.1. To begin with the bias is disregarded. From this analysis it isn't difficult to generalise to the general case.

The weight $w_{i,j}$ is the number that the input, in the $j$:th place, are multiplied with, before this value is used as one of the inputs of the $i$:th neuron. If there aren't any connections the weights are zero between these nodes. A summation is made in for instance neuron $i$, in the first hidden layer, of all the inputs, that are multiplied with non zero weights, and that are connected with neuron $i$. A new input ($h_i^{(\mu)}$) is then obtained, which is used as the input of the $i$:th neuron. Expressed in formulas ($h_i^{(\mu)}$) is obtained in the following way:

$$h_i^{(\mu)} = \sum_{j=1}^{N_j} w_{i,j} \xi_j^{(\mu)}$$  \hspace{1cm} (2.5)
Where \( N_k \) is the number of connections between the inputs and the neuron \( i \). The index \( k \) tells us for which layer the formula is valid (in the formula above \( k = 1 \)).

Fig 2.3.1. A typical neural network is shown. Most neural networks are much larger, have more neurons in each layer, than this one. Most of the variables in the figure are either going to be explained or are already explained in the text. The bias are explained in section 2.4. There is always only one output layer, but in the general case, it’s possible to have a neural network with more hidden layers.

The neurons in the first (hidden) layer contain a function \( g \) that operates on the inputs, see figure 2.3.1. The inputs to neuron \( i \) (\( h_i^{(\mu)} \)) transforms to a output (\( H_j^{(\mu)} \)) in the following way:

\[
H_j^{(\mu)} = g(h_i^{(\mu)}) = g\left(\sum_{j=1}^{N_k} w_{ij} \xi_j^{(\mu)}\right)
\]  

(2.6)

The inputs, \( \xi_i^{(\mu)} \), to the neurons in the output layer are defined as:

\[
\xi_i^{(\mu)} = \sum_{j=1}^{N_k} W_{ij} H_j^{(\mu)} = \sum_{j=1}^{N_k} W_{ij} g(h_j^{(\mu)}) = \sum_{j=1}^{N_k} W_{ij} g\left(\sum_{k=1}^{N_k} W_{jk} \xi_k^{(\mu)}\right)
\]  

(2.7)
The outputs $H^{(a)}_i$ from the first hidden layer are the inputs to the next layer, which is the output layer in this analysis. The output from neuron $i$ ($O^{(a)}_i$) in the output layer is given by the following formula:

$$O^{(a)}_i = g \left( \sum_{j=1}^{N_s} W_{i,j} H^{(a)}_j \right) = g \left( \sum_{j=1}^{N_s} W_{i,j} g \left( \sum_{k=1}^{N_s} w_{j,k} S^{(a)}_k \right) \right) \quad (2.8)$$

To continue the analysis, it’s necessary to introduce the cost function $E$. It’s often based on a least mean square method and this method is used in this analysis. The cost function $E^{(a)}$ for the input pattern $\mu$ is defined in the following way:

$$E^{(a)} = \frac{1}{2} \sum_{i=1}^{N_a} \left( C^{(a)}_i - O^{(a)}_i \right)^2 \quad (2.9)$$

Note that $C^{(a)}_i$ is the correct output from neuron $i$, when the input pattern $\mu$ is used. The inputs, the outputs that are received and the correct outputs in (2.9) have the superscript $\mu$. This means that all these terms belongs to the input pattern $\mu$. To get the true cost function $E$, it’s necessary to sum the cost functions $E^{(a)}$ over all available patterns $\mu$ in the following way:

$$E = \sum_{\mu=1}^{N_a} E^{(a)} = \frac{1}{2} \sum_{\mu=1}^{N_a} \sum_{i=1}^{N_a} \left( C^{(a)}_i - O^{(a)}_i \right)^2 =$$

$$= \frac{1}{2} \sum_{\mu=1}^{N_a} \sum_{i=1}^{N_a} \left[ C^{(a)}_i - g \left( \sum_{j=1}^{N_a} W_{i,j} g \left( \sum_{k=1}^{N_s} w_{j,k} S^{(a)}_k \right) \right) \right]^2 \quad (2.10)$$

Now when an explicit expression for the cost function $E$ is obtained, the gradient descent method can be used. The gradients of $E$ is multiplied with $(-\eta)$ and the expressions for the changes in the weights $W_{i,j}$ and $w_{i,j}$ is then obtained. Let’s name them $\Delta W_{i,j}$ and $\Delta w_{i,j}$. The correct outputs $C^{(a)}_i$ are constants in these calculations. The variable $\eta$ is the step length. The best strategy for these calculations is to begin with the changes in the weights in the output layer ($\Delta W_{i,j}$’s), because they are easiest to calculate. By using the chain rule the following formula for these changes in the weights are obtained:
\[
\Delta W_{i,j} = \sum_{\mu=1}^{N_\mu} \Delta W_{i,j}^{(\mu)} = -\eta \frac{\partial E}{\partial W_{i,j}} = -\eta \frac{\partial E}{\partial \alpha_{i,j}} \frac{\partial \alpha_{i,j}}{\partial W_{i,j}} = -\eta \sum_{\mu=1}^{N_\mu} \frac{\partial C_{i,j}^{(\mu)}}{\partial W_{i,j}} \frac{\partial \alpha_{i,j}^{(\mu)}}{\partial W_{i,j}} =
\]

\[
= \eta \sum_{\mu=1}^{N_\mu} \left( C_{i,j}^{(\mu)} - O_{i,j}^{(\mu)} \right) g(\zeta_{i,j}^{(\mu)}) g(h_{i,j}^{(\mu)})
\]

(2.11)

As can be seen from the formula (2.11), the changes in the weights are calculated for each pattern \(\mu\) (\(\Delta W_{i,j}^{(\mu)}\)) and are then summed over all available patterns (which gives \(\Delta W_{i,j}\)). Note that it’s possible to update the changes of the weights, \(W_{i,j}^{(\mu)}\), directly with \(\Delta W_{i,j}^{(\mu)}\) for each pattern \(\mu\). All weights and biases may be updated in this way, see formula (2.30)-(2.33). This is done in the neural networks in chapter 4 and chapter 5. Now a new variable called \(\delta_i^{(\mu)}(o)\) is defined, because it’s then easier to compare the formulas for \(\Delta W_{i,j}\) and \(\Delta w_{i,j}\). Observe that the \(o\) in \(\delta_i^{(\mu)}(o)\) means that these \(\delta\)'s are in some way connected to the output layer. These \(\delta\)'s are also connected to the pattern \(\mu\). These variables are defined in the following way:

\[
\delta_i^{(\mu)}(o) = g'(\zeta_i^{(\mu)}) \left[ C_i^{(\mu)} - O_i^{(\mu)} \right]
\]

(2.12)

It’s now possible to write (2.11) in the following way:

\[
\Delta W_{i,j} = \eta \sum_{\mu=1}^{N_\mu} \delta_i^{(\mu)}(o) H_j^{(\mu)}
\]

(2.13)

The remaining changes in the weights (\(\Delta w_{i,j}\)) are connected with the first (hidden) layer. These calculations are of course also based on the gradient descent method. To make the calculations easier, the chain rule is used. After some straightforward calculations the following expressions are obtained for these changes in the weights:

\[
\Delta w_{i,j} = \sum_{\mu=1}^{N_\mu} \Delta w_{i,j}^{(\mu)} = -\eta \frac{\partial E}{\partial W_{i,j}} = -\eta \sum_{\mu=1}^{N_\mu} \frac{\partial C_{i,j}^{(\mu)}}{\partial W_{i,j}} \frac{\partial \alpha_{i,j}^{(\mu)}}{\partial W_{i,j}} =
\]

\[
= \eta \sum_{\mu=1}^{N_\mu} \left( C_{i,j}^{(\mu)} - O_{i,j}^{(\mu)} \right) g'(\zeta_{i,j}^{(\mu)}) W_{k,j} g'(h_{i,j}^{(\mu)}) \xi_j^{(\mu)} =
\]

\[
= \eta \sum_{\mu=1}^{N_\mu} \delta_i^{(\mu)}(o) W_{k,j} g'(h_{i,j}^{(\mu)}) \xi_j^{(\mu)} = \eta \sum_{\mu=1}^{N_\mu} \delta_i^{(\mu)}(h) \xi_j^{(\mu)}
\]

(2.14)

Where \(\Delta w_{i,j}^{(\mu)}\) are the changes in the weights given the input pattern \(\mu\). A summation over all input patterns is made in the formula (2.14). This summation is done in exactly the same way as in formula (2.11). In formula (2.14) a new kind of \(\delta\) is defined, \(\delta_i^{(\mu)}(h)\). The \(h\) in the variable tells us, that it's in some way connected to the first (hidden) layer, and the superscript \(\mu\) tells us, to which input pattern it belongs. According to the formula (2.14), it's defined in the following way:
\[ \delta^{(\mu)}_i(h) = g\left(h^{(\mu)}_i \right) \sum_{k=1}^{N_k} W_{ik} \delta^{(\mu)}_k(o) \] \hfill (2.15) 

From the analysis above, some underlying structures can be seen. All the formulas, that calculates the changes in the weights, can in the general case be written in the following way:

\[ \Delta W_{p,q} = \eta \sum_{\text{patterns}} \delta_{\text{output-signals}} H_{\text{output-signals}} \] \hfill (2.16)

Where the output signals refers to the signals, that are outputs from the neurons in a certain layer, and the input signals are the inputs, to the neurons in the same layer. The \( \delta^{(\mu)}_i \)'s are closely related to the difference between the correct (desired) output and the output, that is received from the neural network.

As can be seen from the analysis above, it's easy to extend this analysis to the case when there are more layers and when the neural network is allowed to have different transfer functions in the different layers. It is only necessary to modify the analysis, in this chapter, a bit. The formulas become more complex and it is necessary to make more use of the chain rule, when calculations must be made for the more and more complex expressions for the gradients of \( E \) in the different layers.

### 2.4 Bias

It isn’t difficult to extend this analysis to the case, when also biases are used. In most of the formulas in section 2.3, it is only necessary to make small changes. All the definitions that already are made, are also going to apply here. The only differences are that the bias terms are added in the right places in these formulas. Using the same names on the variables as in section 2.3, the formulas (2.5)-(2.10) are changed to the following formulas:

\[ h^{(\mu)}_i = \sum_{j=1}^{N_j} w_{ij} s^{(\mu)}_j + b_i \] \hfill (2.17)

\[ H^{(\mu)}_i = g\left(h^{(\mu)}_i \right) = g\left(\sum_{j=1}^{N_j} w_{ij} s^{(\mu)}_j + b_i \right) \] \hfill (2.18)

\[ z^{(\mu)}_i = \sum_{j=1}^{N_j} W_{ij} H^{(\mu)}_j + B_i = \sum_{j=1}^{N_j} W_{ij} g\left(\sum_{k=1}^{N_k} w_{jk} s^{(\mu)}_k + b_j \right) + B_i \] \hfill (2.19)

\[ O^{(\mu)}_i = g\left(z^{(\mu)}_i \right) = g\left(\sum_{j=1}^{N_j} W_{ij} s^{(\mu)}_j + B_i \right) \] \hfill (2.20)
\[ E^{(\mu)} = \frac{1}{2} \sum_{i=1}^{N_2} \left( C_i^{(\mu)} - O_i^{(\mu)} \right)^2 = \]

\[ = \frac{1}{2} \sum_{i=1}^{N_2} \left[ C_i^{(\mu)} - g \left( \sum_{j=1}^{N_2} W_{i,j} g \left( \sum_{k=1}^{N_4} W_{j,k} \zeta_k^{(\mu)} + b_j \right) + B_i \right) \right]^2 \]  

(2.21)

\[ E = \sum_{\mu=1}^{N_4} E^{(\mu)} = \frac{1}{2} \sum_{\mu=1}^{N_4} \sum_{i=1}^{N_2} \left[ C_i^{(\mu)} - O_i^{(\mu)} \right]^2 = \]

\[ = \frac{1}{2} \sum_{\mu=1}^{N_4} \sum_{i=1}^{N_2} \left[ C_i^{(\mu)} - g \left( \sum_{j=1}^{N_2} W_{i,j} g \left( \sum_{k=1}^{N_4} W_{j,k} \zeta_k^{(\mu)} + b_j \right) + B_i \right) \right]^2 \]  

(2.22)

Where \( b_i \) is the bias that is added to neuron \( i \) in the first (hidden) layer and \( B_i \) is the bias that is added to neuron \( i \) in the output layer. All other variables, indices and superscripts has already been explained in section 2.3 and are therefore not repeated here. The formulas (2.11)-(2.16) look exactly as they did in section 2.3, if the new definitions of the variables \( h_i^{(\mu)} \), \( H_i^{(\mu)} \), \( \zeta_i^{(\mu)} \), \( O_i^{(\mu)} \), \( E^{(\mu)} \) and \( E \) are used. These variables are in this case (when the bias is used) defined by the formulas (2.17)-(2.22). The derivations and the analysis of the new formulas are of course the same as in the preceding section.

The gradient descent method is used to derive the formulas for the changes of the bias (\( \Delta b_i \) and \( \Delta B_i \)). It is then necessary to calculate the gradients of \( E \) with respect to \( b_i \) and \( B_i \). The calculations and the analysis are analogous with the case, when the formulas for the changes in weights are derived. Also here the chain rule is used. The changes, that are related to the output layer, are derived first, because these variables are easiest to calculate. These changes in the bias (the \( \Delta b_i \)'s) are obtained in the following way:

\[ \Delta B_i = \sum_{\mu=1}^{N_4} \Delta B_i^{(\mu)} = -\eta \frac{\partial E}{\partial B_i} = -\eta \frac{\partial E}{\partial O_i} \frac{\partial O_i}{\partial \zeta_i} \frac{\partial \zeta_i}{\partial \zeta_i} = \]

\[ = \eta \sum_{\mu=1}^{N_4} \left( C_i^{(\mu)} - O_i^{(\mu)} \right) g' \left( \zeta_i^{(\mu)} \right) \]  

(2.23)

By using the chain rule once more the changes of the bias in the first (hidden) layer (the \( \Delta b_i \)'s) are obtained in the following way:

\[ \Delta b_i = \sum_{\mu=1}^{N_4} \Delta b_i^{(\mu)} = -\eta \frac{\partial E}{\partial b_i} = -\eta \sum_{\mu=1}^{N_4} \frac{\partial E}{\partial h_i^{(\mu)}} \frac{\partial h_i^{(\mu)}}{\partial b_i} = \]

\[ = \eta \sum_{\mu=1}^{N_4} \sum_{k=1}^{N_2} \left( C_k^{(\mu)} - O_k^{(\mu)} \right) g' \left( \zeta_k^{(\mu)} \right) W_{i,k} g \left( h_i^{(\mu)} \right) \]  

(2.24)
Observe that in the formulas (2.23) and (2.24) it is necessary to use the new definitions, which are made in this section (the formulas (2.17)-(2.22)).

As in section 2.3 $\delta$'s can be defined, to enable us to more easily see the connections between the different formulas. Since the $\delta$'s, that has to do with the changes in weights already have been taken care of, it is only necessary to define new $\delta$'s for the changes in bias. They are defined in the following way:

\[
\delta_i^{(\mu)}(B) = \left( c_i^{(\mu)} - o_i^{(\mu)} \right) g' \left( \zeta_i^{(\mu)} \right) \tag{2.25}
\]

\[
\delta_i^{(\mu)}(b) = \sum_{k=1}^{N_k} \left( c_k^{(\mu)} - o_k^{(\mu)} \right) g' \left( \zeta_k^{(\mu)} \right) w_{ki} g' \left( h_i^{(\mu)} \right) \tag{2.26}
\]

Where the $b$'s and the $B$'s tell us that these variables are in some way connected to the bias. The $B$ in one of the variables tells us that this variable is in some way connected to the output layer and in the same way will the $b$ in the other variable tell us that it's in some way connected to the first (hidden) layer. The indices and superscripts has already been explained in section 2.3 and are therefore not repeated here. These newly defined $\delta$'s enable us to write the formulas (2.23) and (2.24) in the following way:

\[
\Delta B_i = \eta \sum_{\mu=1}^{N_k} \delta_i^{(\mu)}(B) \tag{2.27}
\]

\[
\Delta b_i = \eta \sum_{\mu=1}^{N_A} \sum_{k=1}^{N_k} \delta_i^{(\mu)}(B) w_{ki} g' \left( h_i^{(\mu)} \right) \tag{2.28}
\]

To make the formulas (2.25)-(2.28) valid, it is necessary to use all the definitions for the different variables from this section (in other words only the definitions of the variables, which are obtained from the formulas (2.17)-(2.24), must be used).

It is possible to have different step lengths in the backpropagation algorithm. In the bias it's possible to have $\eta_i^b$ in the $\Delta b_i$ term and $\eta_i^o$ in the $\Delta B_i$ term, instead of only using $\eta$ in all these expressions. It's of course possible to do the same thing with the changes in weights. In other words it's possible to have $\eta_{ij}^b$ in the $\Delta w_{ij}$ term and $\eta_{ij}^o$ in the $\Delta W_{ij}$ term.

It's also possible to have an adaptive step length. The step length is an important parameter. If the step length is too large the system can become unstable and the errors from the cost function can be increasing, instead of decreasing. If it's only a little bit too large an oscillation takes place around the minimum, which never is reached. If it's too small, the time until it reaches the minimum becomes extremely long. If only the step length is small enough, a local minimum (which sometimes also is the global minimum) is always reached. To know how the step length must be changed, the cost function (error function) is studied. If the error function $E$ has increased, while a step is taken, a too large step is used. The step length is then decreased by the factor $1 - \delta$, where $\delta$ has a small positive value, for instance 0.05. Note that it's possible to take a step in the opposite direction to undo the wrong step. If the error
function has decreased ($\Delta E < 0$), a larger step can be used. The step length is then multiplied by the factor $1 + a$, where $a$ is a small positive value, for instance 0.05 and a step is taken in the usual direction. In most cases the factor $d$ decreases the step length $\eta$ more than $a$ increases it. The following formula is obtained for the new step length:

$$
\eta(t + 1) = \begin{cases} 
\eta(t) + \eta(t)a & \text{when } \Delta E < 0 \\
\eta(t) - \eta(t)d & \text{when } \Delta E > 0 \\
\eta(t) & \text{when } \Delta E = 0 
\end{cases}
$$

(2.29)

Where $\eta(t + 1)$ is the new value of the step length, $\eta(t)$ is the preceding value of the step length and $\Delta E$ is the change in the error function during this step.

Now that all expressions that are needed have been derived, it’s possible to update the neural network in the following way:

$$
W_{i,j}^{\text{NEW}} = W_{i,j}^{\text{OLD}} + \Delta W_{i,j} = W_{i,j}^{\text{OLD}} + \sum_{\mu=1}^{N_{\mu}} \Delta W_{i,j}^{(\mu)}
$$

(2.30)

$$
w_{i,j}^{\text{NEW}} = w_{i,j}^{\text{OLD}} + \Delta w_{i,j} = w_{i,j}^{\text{OLD}} + \sum_{\mu=1}^{N_{\mu}} \Delta w_{i,j}^{(\mu)}
$$

(2.31)

$$
B_i^{\text{NEW}} = B_i^{\text{OLD}} + \Delta B_i = B_i^{\text{OLD}} + \sum_{\mu=1}^{N_{\mu}} \Delta B_i^{(\mu)}
$$

(2.32)

$$
b_i^{\text{NEW}} = b_i^{\text{OLD}} + \Delta b_i = b_i^{\text{OLD}} + \sum_{\mu=1}^{N_{\mu}} \Delta b_i^{(\mu)}
$$

(2.33)

This part of the algorithm improves the neural network in the sense, that it enables the neural network to do better generalisations, when similar input patterns, as those it is trained with, are used. As can be seen from the formulas (2.30)-(2.33) it’s easy to update the neural network one step at the time. It’s possible to update each of the variables on the left side of the formulas (2.30)-(2.33) once for each pattern $\mu$. The formulas above simply say that the new values of the weights or bias are obtained from the old values, plus the change they got from the gradient descent method.

### 2.5 Momentum

The advantage with using a momentum term is that the backpropagation algorithm is not going to get stuck in a local minimum, if it is shallow enough. Instead it continues past the local minimum and in the end reaches the global minimum (see fig 2.5.1). There are two different ways to introduce a momentum term in the formulas (2.30)-(2.33). In fig 2.5.1 the momentum term, $\alpha_i$, is used in the following way in formula (2.30):

$$
\Delta W_{i,j}(t + 1) = -(1 - \alpha_i)\eta \frac{\partial E}{\partial W_{i,j}} + \alpha_i \Delta W_{i,j}(t)
$$

(2.34)
This is the first way the momentum term can be used in the changes of weights and bias, see Demuth and Beale (93) [4] p. 5.32-5.36. Similar formulas are of course obtained for the formulas (2.31)-(2.33), when the momentum term, \( \alpha \), is introduced in these formulas.

Note that the remaining material in this section with the exception of fig 2.5.1 is taken from Hertz, Krogh and Palmer (91) [1] p. 123.

The momentum term acts as a filter in the gradient descent method, because the momentum terms contains the old changes of weights or bias. The momentum terms are scaled by the value \( \alpha \) for each step in the updating part of the backpropagation algorithm. The momentum parameter \( \alpha \) has a value between 0 and 1. The older momentum terms are therefore less significant than the newer ones. The formulas (2.30)-(2.33) are changed into the following formulas, when the second way to introduce the momentum term, \( \alpha \), is used:

\[
\Delta W_{ij}(t+1) = -\eta \frac{\partial E}{\partial W_{ij}} + \alpha \Delta W_{ij}(t)
\]

(2.35)

\[
\Delta w_{ij}(t+1) = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t)
\]

(2.36)

Fig 2.5.1. If the momentum term is large enough (in this example the momentum parameter \( \alpha \) is set to 0.95) the learning process do not end in the local minimum, if it is shallow enough. Instead it continues past the local minimum into the global minimum. There it oscillates across the global minimum, until it finally comes at rest. This picture is taken from Demuth and Beale (93) [4] p. 5.36.
\[ \Delta B_i(t+1) = -\eta \frac{\partial E}{\partial B_i} + \alpha \Delta B_i(t) \quad (2.37) \]

\[ \Delta b_i(t+1) = -\eta \frac{\partial E}{\partial b_i} + \alpha \Delta b_i(t) \quad (2.38) \]

The formulas (2.35)-(2.38) show how the momentum term, \( \alpha \), is used in the neural networks in chapter 4 and chapter 5.

Another effect of the momentum term is that once the gradient of \( E \) becomes stabilised to an almost fixed value, a larger learning rate (\( \eta \)) is received after a lot of steps. The new value of \( \eta \), \( \eta^{NEW} \), can then approximately be calculated by the following formula:

\[ \eta^{NEW} = \frac{\eta}{1-\alpha} \quad (2.39) \]

Where the momentum term, \( \alpha \), has been introduced according to the formulas (2.35)-(2.38).
3 Extremal seeking control

In many applications the objective is to maximise or minimise some criteria. It could, for instance, be desirable to produce as much as possible of a special chemical compound. One of the components in this compound could be very expensive and the goal would then be to minimise the use of this component. It’s not difficult to find other examples, where it’s desirable to reach an extremum. In this project the main objective is to locate the extremum and then to track this varying extremum. The process of tracking an extremum is called extremal seeking control. In this case the extremum, that is tracked, is a steady state minimum.

A comparison is made between three different kind of controllers, that are tracking a varying set point. These controllers are the constant extremum controller, the mixed extremum controller and controllers based on neural networks. A more detailed explanation of these controllers will be made further on.

For a system to have an extremum it must be non-linear. It’s then obvious that the output $y$ must be a non-linear function of the control signal $u$ for these kind of systems. The non-linear functions must be appropriately approximated. These approximations are made in such a way that they give a good model of the system and at the same time they shouldn’t make the model so complex, that system analysis is impossible. One way to make these approximations are to use some kind of series expansions of the non-linear functions. Commonly used series expansions are the Volterra, Wiener and Uryson series. In this project a special case of the Uryson series expansion is used, the Hammerstein model. The material in this chapter is taken from Åström and Wittenmark (90) [2], Urquhart (95) [6], Nilsson and Sandberg (93) [7], Wittenmark (93) [8] and Wittenmark and Urquhart (95) [9].

3.1 The Hammerstein model

A typical Hammerstein model in discrete-time, see Urquhart (95) [9] p. 3-5, is based on the following formula:

$$ A^*(q^{-1})y(k) = B^*(q^{-1})f(u(k)) + C^*(q^{-1})e(k) $$  \hspace{1cm} (3.1)

Where $f$ is a non-linear function, $e(k)$ is Gaussian white noise and $A^*(q^{-1})$, $B^*(q^{-1})$ and $C^*(q^{-1})$ are polynomials in the backward shift operator $q^{-1}$. Since models based on extremal seeking control are studied, it’s important that the behaviour of the non-linear functions are correctly modelled close to the extremums. Most non-linear functions are well approximated by a quadratic polynomial near an extremum. If polynomials of higher order or other functions are used in the series expansions the system analysis will be too complex. Therefore the following simplified Hammerstein model is chosen:

$$ A^*(q^{-1})y(k+1) = B_0^* + B_1^*(q^{-1})u(k) + B_2^*(q^{-1})u^2(k) + C^*(q^{-1})e(k+1) $$  \hspace{1cm} (3.2)
The following process (see Wittenmark and Urquhart (95) [9] p. 1642-1643) is studied more closely in this project:

\[ A(q^{-1}) = 1 + a_1q^{-1} = 1 - 0.95q^{-1} \]
\[ B_0 = b_0 = 0.25 \]
\[ B_1(q^{-1}) = b_{10} + b_{11}q^{-1} = 0.5 - 0.25q^{-1} \]
\[ B_2(q^{-1}) = b_{20} = 0.5 \]
\[ C(q^{-1}) = 1 + c_1q^{-1} = 1 - 0.5q^{-1} \]

\[ e(k + 1) = \text{Zero Mean Gaussian White Noise with the Variance } \sigma_e^2 \quad (3.3) \]

All these formulas are named (3.3), so that easier references can be made to this dynamical process. The polynomial \( C(q^{-1}) \) is of special interest. The roots of \( C(q^{-1}) \) are inside the unit circle, so this polynomial is stable in the process (3.3). If the \( C(q^{-1}) \) polynomial is unstable a mirrored polynomial has to be determined, see Åström and Wittenmark (90) [2] p. 368-369.

### 3.2 Control objectives

To compare different controllers an optimisation criteria needs to be formulated. One way to find such a criteria is to use minimum-variance control, see Urquhart (95) [6] p. 5 and Åström and Wittenmark (90) [2] p. 369. The output \( y \) should then follow the desired set point \( y_0 \) as closely as possible. Since the errors, in the processes that are studied, are uncorrelated with zero mean, no integrations are necessary. The processes that are chosen are in discrete time, hence it’s logical to choose a criteria that’s based on a sum, that contains some kind of quadratic errors. In this project the minimum-variance control is simplified to following loss function:

\[ J(u_{\text{Control}}) = \sum_{k=k_0}^{N} (y(k+1) - y_0)^2 \quad (3.4) \]

Where \( u_{\text{Control}} \) is the given control strategy in the loss function \( J \). In the real minimum-variance control an expectation is taken over the quadratic terms in (3.4) and the result of this sum is divided by \( N \). The following formula is then obtained (when \( N \to \infty \)):

\[ J_{\text{mvc,\infty}}(u_{\text{Control}}) = \lim_{N \to \infty} E \left[ \frac{1}{N} \sum_{k=k_0}^{N} (y(k+1) - y_0)^2 \right] \quad (3.5) \]
Where the control strategy $u_{\text{Control}}$ is chosen in (3.5). The name minimum-variance comes from the fact that the variance of the output $y$ is minimised around the set point $y_0$. An extremum is often chosen as the set point. A steady state minimum is chosen as the set point in the controllers that are compared in this project. In simulations of the controllers the process parameters are known and thus the steady state minimum $y_0$, see formula (3.9).

### 3.3 Admissible controls

A control law is admissible, if the control signal $u(k)$ is a function of earlier control signals $u$ and outputs $y$ up to time $k$. A set $Y_k$ is defined for these control signals and outputs, that are available at time $k$. This set is defined in the following way:

$$Y_k = [y(k), y(k-1), ...; u(k-1), u(k-2), ...] \in \mathbb{R}$$

(3.6)

Where the control law $u(k) \in Y_k$, which means that the control $u(k)$ can be calculated from the set $Y_k$, that consist of real signals. Note that from formula (3.6) it's possible to draw the conclusion that the control signals are causal.

### 3.4 The constant extremum controller

It's assumed that the controllers are going to track a varying minimum. The constant extremum controller, see Wittenmark and Urquhart (95) [9] p. 1640-1642 and Wittenmark (93) [8] p. 331-334, is the simplest controller, that is used to solve this problem, in the simulations. Already in the name of this controller, the fact that a constant control law, $u(k) = u_0$, is used, can be seen. The idea behind this controller is that the right steady state minimum of $y$, $y_0$, is obtained for the process. The process, given by (3.2) and (3.3), is studied in steady state and the errors, $e(k)$, are set to zero, before the calculation of the minimum or the set point, $y_0$, is done. This set point corresponds to the constant control signal $u_0$, which isn't difficult to calculate once the formula for $y_0$ has been found. The derivations of $u_0$ and $y_0$ can be seen in Wittenmark and Urquhart (95) [9] p. 1649-1651. To make these calculations possible the system must be stable. The steady state minimum or the set point, $y_0$, and the constant control law $u(k) = u_0$ is given by the following formulas (if $B_2 \neq (1)$):

$$u_0 = -\frac{B_1 \ast (1)}{2B_2 \ast (1)}$$

(3.7)

$$y_0 = \frac{1}{A \ast (1)} \left[ B_0 \ast \frac{-B_1 \ast (1)^2}{2B_2 \ast (1)} + \frac{B_1 \ast (1)^2}{4B_2 \ast (1)} \right]$$

(3.8)
The polynomials $A^*(z^{-1})$ and $B_i^*(z^{-1})$ for $i = 0, 1, 2$ are polynomials expressed in the complex variable $z^{-1}$, which corresponds to the backward shift operator $q^{-1}$. This complex variable, $z^{-1}$, is given the value 1 in steady state.

The constant extremum controller is not a very good controller, since the dynamics of the process and the error sequence $\{e(k)\}$ has not been considered. The mixed extremum controller, that is explained later, is a better controller for extremal seeking control. For the dynamical process that is chosen in the simulations, the process given by the formulas (3.2) and (3.3), the following values of $y_0$ and $u_0$ are received:

$$\begin{align*}
y_0 &= 4.375 \\
u_0 &= -0.25
\end{align*} \quad (3.9)$$

3.5 The mixed extremum controller

The mixed extremum controller consists of two controllers. A minimum variance controller is used, when it's possible and when it isn't possible to use a minimum variance controller, the constant extremum controller is used, see Wittenmark and Urquhart (95) [9] p. 1640-1642.


The theory behind the polynomial identity, given by formula (3.10), and minimum-variance control can be seen in Wittenmark (93) [8] p. 328-335 and Åström and Wittenmark (90) [2] p. 160-163, 169, 180-181 and 366-384. The following polynomial identity is introduced (see Wittenmark (93) [8] p. 329):

$$C^*(q^{-1}) = A^*(q^{-1})F^*(q^{-1}) + q^{-m}G^*(q^{-1}) \quad (3.10)$$

Where all the polynomials are in the backward shift operator $q^{-1}$. The $C^*(q^{-1})$ polynomial and the $A^*(q^{-1})$ polynomial are given by the process model. The $F^*(q^{-1})$ polynomial, which is of order $m-1$, is related to the prediction error and the $G^*(q^{-1})$ polynomial is related to the $m$-step prediction. The fact that $A^*(q^{-1})$ is a polynomial of the same order as $C^*(q^{-1})$ in the one step predictor, can be used in the polynomial division given by (3.10). From formula (3.10) it's then obvious that $F^* = 1$ and $m = 1$ for the one step predictor. By using the polynomial division (3.10), for the one step predictor, in (3.2) the following result is obtained:

$$\begin{align*}
(C^* - q^{-1}G^*)y(k+1) &= B_0^* + B_1^* u(k) + B_2^* u^2(k) + C^* e(k+1) \\
y(k+1) &= \frac{1}{C^*} \left( B_0^* + B_1^* u(k) + B_2^* u^2(k) + G^* y(k) \right) + e(k+1) \quad (3.11)
\end{align*}$$
Where \( q^{-1} \) terminology has been suppressed in the polynomials for simplicity. The one step predictor is easily identified from (3.11). The following formula is then obtained for the one step predictor:

\[
\hat{y}(k+1 \mid Y_k) = \frac{1}{C^*}(B_0 + B_1 u(k) + B_2 u^2(k) + G^* y(k))
\]  
(3.12)

Where the one step predictor, \( \hat{y}(k+1 \mid Y_k) \), is a function of the admissible signals, given by the set \( Y_k \), see formula (3.6), and the control signal \( u(k) \). The prediction error is given by \( e(k+1) \). For the system and process given by (3.2) and (3.3), the one step predictor in (3.12) becomes:

\[
\hat{y}(k+1 \mid Y_k) = \frac{1}{C^*}(b_0 + b_{10} u(k) + b_{11} u(k-1) + b_{20} u^2(k) + (c_1 - a_1) y(k))
\]  
(3.13)

The criteria that is minimised in a minimum variance controller is given by:

\[
V_1 = E\left[ \left( y(k+1) - y_0 \right)^2 \right] = E\left[ \left( \hat{y}(k+1 \mid Y_k) - y_0 \right)^2 + [e(k+1)]^2 \right] =
\]

\[
E\left[ \left( \hat{y}(k+1 \mid Y_k) - y_0 \right)^2 \right] + E[e^2(k+1)] =
\]

\[
E\left[ \left( \hat{y}(k+1 \mid Y_k) - y_0 \right)^2 \right] + \sigma_e^2 =
\]

\[
E[p^2(k+1)] + \sigma_e^2
\]  
(3.14)

It’s assumed that only the control signal \( u(k) \) and signals in the set \( Y_k \), given by (3.6), can be used in the one step predictor. This assumption implies that nothing can be done to reduce the variance of \( e(k+1) \). In the first equality formula (3.11) and (3.12) are used. Note that the prediction error, \( e(k+1) \), is independent of the signals in the set \( Y_k \), given by (3.6). This set contains all the signals that exist in the one step predictor, \( \hat{y}(k+1 \mid Y_k) \). Hence the conclusion, that the prediction error is independent of the one step predictor, can be drawn. The admissible signals in the set \( Y_k \), given by (3.6), and the control signal \( u(k) \) are the available signals at time \( k \), that are used by the one step predictor. The fact that \( e(k+1) \) is independent of the one step predictor makes the second equality to hold. The variance \( \sigma_e^2 \) is a constant. It’s the variance of the error \( e(i) \) for any \( i \). Remember that \( \{e(k)\} \) is a sequence of independent random variables with the zero mean and the standard deviation \( \sigma_e \). From formula (3.14) it possible to draw the conclusion that the same results are obtained if the variance of \( p(k+1) = \left( \hat{y}(k+1 \mid Y_k) - y_0 \right) \) is minimised instead of the variance of
\((y(k+1) - y_0)\). The variable \(y_0\) is the set point, given by (3.8) and (3.9). The new minimum variance criteria is now given by:

\[
V = E\left[\left(j(k+1 \mid Y_k) - y_0\right)^2\right] = E[p^2(k+1)] = p^2(k+1) \tag{3.15}
\]

Where \(p(k+1)\) is defined by:

\[
p(k+1) = j(k+1 \mid Y_k) - y_0 \tag{3.16}
\]

All variables in the formulas, given by (3.15) and (3.16), are of course defined in the same way as they are in formula (3.14). It’s possible to interpret \(p(k+1)\) as the prediction of \((y(k+1) - y_0)\). To make it easier to find the control law, given by \(u(k)\), for the mixed extremum controller, \(p(k+1)\) is separated into two parts. The first part contains a quadratic function of \(u(k)\) with the exclusion of the constant term and the second part contains what’s left of \(p(k+1)\). The second part is called \(w(k)\). The term \(p(k+1)\) is divided into the two parts discussed above, see Wittenmark and Urquhart (95) [9] p. 1641-1643. The following expression is obtained for \(p(k+1)\):

\[
p(k+1) = b_{10}u(k) + b_{20}u^2(k) + w(k) \tag{3.17}
\]

For the process and system model given by (3.2) and (3.3) the following formulas are obtained for \(w(k)\):

\[
w(k) = \frac{1}{C_{\ast}}[b_0 + (b_{11} - c_1b_{10})u(k-1) + (-c_1b_{20})u^2(k-1) + (c_1 - a_1)y(k)] - y_0 \tag{3.18}
\]

Note that the function \(w(k)\) only contains the admissible signals in the set \(Y_k\), given by (3.6).

If \(V\) in formula (3.15) is minimised with respect to \(u(k)\), the one step controller is obtained. The one step controller, see Wittenmark (93) [8] p.329-333, is given by (where it’s assumed that \(b_{20} > 0\)):

\[
u(k) = -\frac{b_{10}}{b_{20}} \quad \text{if } w(k) \geq \frac{b_{10}^2}{4b_{20}} \tag{3.19a}
\]

\[
u(k) = -\frac{b_{10}}{b_{20}} \pm \sqrt{\frac{b_{10}^2}{b_{20}^2} - \frac{w(k)}{b_{20}}} \quad \text{if } w(k) < \frac{b_{10}^2}{4b_{20}} \tag{3.19b}
\]

20
Three solutions are obtained for the one step controller, see formula (3.19a) and (3.19b). Note that the solutions, that are given by (3.19b), exists if and only if \( w(k) \leq \frac{b_{10}^2}{4b_{20}} \). It’s of course assumed that only real roots are acceptable in the solution. The other solution, given by (3.19a), can be identified as a constant controller. The constant controller given by (3.19a) is used, when the minimum-variance control law given by (3.19b) can’t be used.

If the minimum-variance controller, given by (3.19b), can be used, the \( u(k) \) is chosen so that \( p(k+1) = 0 \). The value of \( y(k+1) \) is then given by:

\[
y(k+1) = y_0 + e(k+1)
\]

(3.20)

From formula (3.20) it’s evident that the variance \( V_1 \) in (3.14) becomes:

\[
V_1 = E\left[ (y(k+1) - y_0)^2 \right] = \sigma^2
\]

(3.21)

It’s the lowest variance \( V_1 \) that is obtainable. When the constant controller, given by (3.19a), must be used, the variance \( V_1 \) in (3.14) is always higher, than the variance \( V_1 \) that is obtained, when the minimum-variance controller may be used. A better constant controller for the minimum variance problem, given by minimising \( V_1 \) in (3.14), is the constant extremum controller. Note that all the calculations, that has been done in this section, assume that the system is stable. This fact implies that the constant extremum controller is the best constant controller for the minimum variance problem, see section 3.4. It’s also assumed that the extremum is a steady state minimum, see formula (3.7)-(3.9). If the constant controller, given by (3.19a), is exchanged with the constant extremum controller, given by (3.7), the mixed extremum controller is obtained. The mixed extremum controller is given by:

\[
\left\{ \begin{array}{ll}
  u(k) &= -\frac{B_2^*}{2B_1^*} \frac{(1)}{k} & \text{if } w(k) \geq \frac{b_{10}^2}{4b_{20}} \\
  u(k) &= -\frac{b_{10}}{b_{20}} \pm \sqrt{\left( \frac{b_{10}}{b_{20}} \right)^2 - \frac{w(k)}{b_{20}}} & \text{if } w(k) < \frac{b_{10}^2}{4b_{20}} 
\end{array} \right.
\]

(3.22)

Where \( B_i^* \), for \( i = 1 \) and \( 2 \), are the backward shift polynomials in the process (3.2) with the complex variable \( z^{-1} = 1 \), see also section 3.4. It’s of course assumed that \( b_{20} > 0 \) in the solution given by (3.22). The variance \( V_1 \) given by (3.14), when the constant extremum controller must be used in (3.22), is larger than the variance \( V_1 \), when the minimum variance controller given by (3.19b) may be used. It’s therefore desirable to choose the \( u(k) \) in (3.19b), so that the minimum variance controller may be used as often as possible. From (3.22) it can be seen that \( w(k) \) should be as small as possible, to be able to use the solution given by (3.19b) as often as possible. If the variance \( V_1 \) in (3.14) is summed over \( k \), it can be
seen that this summed variance has the lowest value, when the \( u(k) \) in (3.19b) is chosen, so that \( w(k) \) is as small as possible. The summed variance that is mentioned above is given by:

\[
V_{\text{sum}} = \sum_{k=k_0}^{N_k} E\left[ (y(k+1) - y_0)^2 \right] = \sum_{k=k_0}^{N_k} \left( E\left[ p^2(k+1) \right] + \sigma_e^2 \right) = \\
= \sum_{k=k_0}^{N_k} \left( p^2(k+1) + \sigma_e^2 \right) \tag{3.23}
\]

Where formula (3.14) is used to derive (3.23) and \( k_0 \) is related to the start value of \( k \). The minimum variance criteria given by (3.23) makes the analysis easier to follow. It’s desirable to choose \( u(k) \), so that \( w(k) \) becomes as small as possible. For this reason the predicted value of \( w(k+1) \) is studied. How the predicted value of \( w(k+1) \) is derived can be seen in Wittenmark (93) [8] p. 329-336 and Wittenmark and Urquhart (95) [9] p. 1641-1643. By inserting the values of the polynomials given by the process (3.3), the following formula is obtained for \( \hat{w}(k+1 \mid Y_k) \):

\[
\hat{w}(k+1 \mid Y_k) = \frac{1}{C_*} \left( b_0 - (1 + c_1) y_0 + (b_{11} - c_1 b_{10}) u(k) - c_1 b_{20} u^2(k) + \\
+ (c_1 - a_i) \bar{y}(k+1 \mid Y_k) \right) \tag{3.24}
\]

Where \( \hat{w}(k+1 \mid Y_k) \) is the predicted value of \( w(k+1) \) and \( \bar{y}(k+1 \mid Y_k) \), given by (3.13), is the predicted value of \( y(k+1) \). In both predictions it’s assumed that the set \( Y_k \), given by (3.6), is the known signals at time \( k \). The set point is of course given by \( y_0 \). The term \( (b_{11} - c_1 b_{20}) \) is zero for the known process given by (3.3). Note that \( \hat{w}(k+1 \mid Y_k) \) is a function of \( u(k) \), only by the term \(-c_1 b_{20} u^2(k) \). Using the values in the process given by (3.3) it’s evident that \(-c_1 b_{20} u^2(k) \geq 0 \). Since it’s desirable to have as small value as possible of \( \hat{w}(k+1 \mid Y_k) \), the magnitude of \( u(k) \) should be chosen as small as possible. It’s accomplished by choosing the plus sign in the solution (3.19b), which is the solution to the minimum variance problem. The deductions, from the analysis given here, is easily seen from (3.22). The plus sign in (3.19b) gives the best one step controller and mixed extremum controller, when the process is given by (3.3), see Wittenmark (93) [8] p. 335. Therefore the plus sign in (3.19b) is chosen, when the mixed extremum controller is used in the simulations.

The reason, why the minimum variance controller can’t be used all the time, is studied. The mixed extremum controller is studied to highlight this problem. The problems arises from the fact that the error sequence \( \{e(k)\} \) may increase the value of \( y(k) \) for different \( k \). Remember that the output \( y(k) \) contains the measure noise \( e(k) \) and earlier measure noise, which is easily seen from the system model (3.2). The situation when \( y(k) \) lies close to the set point \( y_0 \) or slightly above it is first studied. If the noises make the predicted value, \( \bar{y}(k+1 \mid Y_k) \), to
be above the set point, $y_0$, there is not much that can be done to push down the predicted value to the set point. In this case the predicted value $\hat{y}(k + 1 | Y_k)$ can't reach the set point, hence the predicted value always lies above the set point. The constant extremum controller can only choose the $u(k) = u_0$ control law, that minimises $\hat{y}(k + 1 | Y_k)$ with respect to $u(k)$. Nothing can be done to the too large output values, $y(i)$ where $i \leq k$. The noise in $y(i)$ where $i \leq k$ makes the process uncontrollable in the sense that a minimum variance controller can't be used. The minimum variance controller can only be used again, if the noise sequence $\{e(k)\}$ pushes down the predicted values of $y$. Once the predicted value $\hat{y}(k + 1 | Y_k)$ lies below the set point, the minimum variance controller can push it up to the set point. It's because the control law $u(k)$ in (3.19b) can be used, to push up the predicted value $\hat{y}(k + 1 | Y_k)$ to the set point, $y_0$.

![Graph 3.5.1](image)

**Graph 3.5.1.** The control law $u(k)$ as a function of $w(k)$ is plotted for the mixed extremum controller given by (3.25). When $w(k) > 0.125$ the constant extremum controller is used with the constant control law, $u(k) = -0.25$. If $w(k) \leq 0.125$ the minimum variance controller is used.

The minimum variance controller is given by the solution (3.19b), which is the non constant solution of $u(k)$ in (3.25). Two solutions are obtained from the minimum variance controller. The first solution in (3.19b) has a plus sign (the upper curve in graph 3.5.1) and the second solution in (3.19b) has a minus sign (the lower dashed curve in graph 3.5.1). The solution given by the plus sign in (3.25) is used in the simulations, when the mixed extremum controller is used in chapter 4 and chapter 5.
Inserting the values for the system model and process given by (3.2) and (3.3), the mixed extremum controller (3.22), the one step controller (3.19) and the set point \( y_0 \) become:

The mixed extremum controller:

\[
\begin{align*}
    u(k) &= -0.25 & \text{if } w(k) \geq 0.125 \\
    u(k) &= -0.5 \pm \sqrt{0.25 - 2w(k)} & \text{if } w(k) < 0.125
\end{align*}
\]  

(3.25)

The one step controller:

\[
\begin{align*}
    u(k) &= -0.5 & \text{if } w(k) \geq 0.125 \\
    u(k) &= -0.5 \pm \sqrt{0.25 - 2w(k)} & \text{if } w(k) < 0.125
\end{align*}
\]  

(3.26)

The set point \( y_0 \):

\[ y_0 = 4.375 \]  

(3.27)
4 Neural networks in Matlab

Since the Neural Network Toolbox, see Demuth and Beale (93) [4], couldn't be used for the controllers based on neural networks, I had to do my own neural networks. Therefore it's necessary to explain how these controllers and neural networks are implemented in Matlab. In Matlab it's important that matrix operations are used instead of for loops. The reasons for this are that the calculations in Matlab then are faster and the main structures in programmes are easier to follow. A lot of the results in this chapter are based on the results from chapter 3 and especially from chapter 2, why it's recommended to keep an eye on these chapters, while reading the analysis in this chapter. Note that the one step prediction, \( \hat{y}(k + 1 \mid Y_k) \), given by (3.13) is renamed in this chapter to \( \hat{y}(k + 1) \). A lot of the material in this chapter is taken from chapter 2, chapter 3 and Persson and Böiers (88) [3].

The idea behind the controllers, that are based on neural networks, is to take into account the dynamical effects in the process (3.3). One way to achieve this goal is to use the fact, that the neural network error \( E(k + 1) \) can be seen as a function of \( u(k) \), \( u(k - 1) \) and \( u(k - 2) \), see formula (4.38), formula (4.42) and figure 4.1. In the backpropagation algorithm it's

a) Step \( k - 2 \):

\[
\xi^{(k-2)} \xrightarrow{w_i^{(k-2)}, b_i^{(k-2)}} \text{NN}^{(k-2)} \xrightarrow{} u(k-2) \xrightarrow{\text{The dynamical process (3.3).}} y(k-1), \hat{y}(k-1)
\]

b) Step \( k - 1 \):

\[
\xi^{(k-1)} \xrightarrow{w_i^{(k-1)}, b_i^{(k-1)}} \text{NN}^{(k-1)} \xrightarrow{} u(k-1) \xrightarrow{\text{The dynamical process (3.3).}} y(k), \hat{y}(k)
\]

c) Step \( k \):

\[
\xi^{(k)} \xrightarrow{w_i^{(k)}, b_i^{(k)}} \text{NN}^{(k)} \xrightarrow{} u(k) \xrightarrow{\text{The dynamical process (3.3).}} y(k+1), \hat{y}(k+1)
\]

Figure 4.1. a) The neural network, \( \text{NN}^{(k-2)} \), during step \( k - 2 \). b) The neural network during step \( k - 1 \). c) The neural network during step \( k \). The neural network operates as a controller, which is tuned by the neural network error \( E(k + 1) \). \( E(k + 1) \), which only is needed during the training of the neural network, is easily calculated from the predicted value of \( y(k + 1) \), \( \hat{y}(k + 1) \), see formula (4.38) and figure 4.1.1. \( \hat{y}(k + 1) \) is given by formula (4.42). \( w_i^{(k)} \) and \( b_i^{(k)} \) are the weight matrix and the bias vector in neuron layer \( i \) (\( i = 1, 2 \) or 3) during step \( k \), when the the neural network is trained. \( \xi^{(k)} \) is the input to the neural network during step \( k \) and \( u(k) \) is the neural network output or the control signal during step \( k \).
necessary, for instance, to calculate the following matrices: \( \frac{\partial u(k-1)}{\partial \psi_i^{(k-1)}} \) (for all three
controllers, that are based on neural networks, see formula (4.41), formula (4.43) and figure
4.1) and \( \frac{\partial u(k-2)}{\partial \psi_i^{(k-2)}} \) (for controller 3, see formula (4.43) and figure 4.1). A backpropagation
algorithm, that contains the changes of weights or bias at an earlier step (here step \( k-1 \) or
step \( k-2 \)) at step \( k \), are very difficult to implement in the backpropagation algorithm, that is
given by the Neural Network Toolbox. Therefore I had to do my own neural networks.

4.1 The neural network

\[ f_1 = \text{tansig} \quad f_2 = \text{tansig} \quad f_3 = \text{purelin} \]

Figure 4.1.1. a) The neural network during step \( k \), \( NN^{(k)} \), that is used in the controllers in this chapter, can be
seen. b) Shows the neural network, when it’s used as an controller during step \( k \). The error \( E(k+1) \) adjust the
controller. The main structures for controller 1, controller 2 and controller 3 are shown. Note that the error
\( E(k+1) \) only needs to be calculated, when the neural network is trained. The input, to the neural network and
thus for the controller, is given by \( \xi^{(k)} \) and the output is given by the scalar \( u(k) \). The variables, functions and
matrices in this picture are explained in the text in this chapter.
The input, to the controllers based on neural network, is given by \( \xi^{(k)} \) and the output is given by the scalar \( u(k) \), see figure 4.1.1. Once the control signal \( u(k) \) is calculated, it’s possible to calculate \( y(k + 1) \) and \( \dot{y}(k + 1) \). From the one step prediction, \( \dot{y}(k + 1) \), the error \( E(k + 1) \) can be calculated. This is the error in the neural network that is backpropagated. The superscript \( k \) in the variables states that all the values are received during step \( k \) in the neural network.

**Neuron layer 1:**

\[
H^{(k)} = \mathbf{w}^{(k)}_{1} \xi^{(k)} + b^{(k)}_{1}
\]  
(4.1)

\[
H^{(k)} = f_{1}(h^{(k)}) = \text{tansig}(h^{(k)})
\]  
(4.2)

Where \( \xi^{(k)} \) is given by a \((N_{1} \times 1)\) column vector. \( h^{(k)} \), \( H^{(k)} \) and \( b^{(k)}_{1} \) are given by \((N_{2} \times 1)\) column vectors and \( w^{(k)}_{1} \) is a \((N_{2} \times N_{1})\) matrix. \( w^{(k)}_{1} \) is the weight matrix in the first neuron layer, \( b^{(k)}_{1} \) are the bias in the first neuron layer and the inputs to the neural network is given by \( \xi^{(k)} \). The formula (4.1) and (4.2) correspond to formula (2.17) and (2.18), where the weight matrix \( w \) in (2.17) has been renamed to \( w^{(k)}_{1} \) in (4.1). The transfer function \( f_{1} \) in the first neuron layer is given by:

\[
f_{1}(x) = \text{tansig}(x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}
\]  
(4.3)

Which is the tansigmoid function, given by (2.2), with \( \beta = 1 \). The vectors and matrices given by for instance \( h^{(k)} \), \( H^{(k)} \) and \( w^{(k)}_{1} \) are as follows:

\[
h^{(k)} = \begin{bmatrix}
    h_{1}^{(k)} \\
    h_{2}^{(k)} \\
    \vdots \\
    h_{N_{1}}^{(k)}
\end{bmatrix}
\]  
(4.4)

\[
H^{(k)} = f_{1}(h^{(k)}) = \begin{bmatrix}
    f_{1}(h_{1}^{(k)}) \\
    f_{1}(h_{2}^{(k)}) \\
    \vdots \\
    f_{1}(h_{N_{1}}^{(k)})
\end{bmatrix}
\]  
(4.5)

\[
w^{(k)}_{1} = \begin{bmatrix}
    w_{1}^{(k)}(1,1) & w_{1}^{(k)}(1,2) & \cdots & w_{1}^{(k)}(1,N_{1}) \\
    w_{1}^{(k)}(2,1) & w_{1}^{(k)}(2,2) & \cdots & w_{1}^{(k)}(2,N_{1}) \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{1}^{(k)}(N_{2},1) & w_{1}^{(k)}(N_{2},2) & \cdots & w_{1}^{(k)}(N_{2},N_{1})
\end{bmatrix}
\]  
(4.6)
From (4.4) and (4.5) it can be seen that the function $f_1$ operates on each element of the column vector $H^{(k)}$. It's how a function, that operates on a column vector, is defined. A function that operates on a row vector is defined analogously. From (4.1), (4.2) and (4.4)-(4.6) it's easily seen that (2.17) defines the same matrix equation as (4.1), where $w_1^{(k)} = w$.

Neuron layer 2:

The inputs to this neuron layer is given by $H^{(k)}$, which are the outputs from neuron layer 1. The following formulas are obtained for neuron layer 2:

$$\zeta^{(k)} = w_2^{(k)} H^{(k)} + b_2^{(k)} \quad (4.7)$$

$$\gamma^{(k)} = f_2(\zeta^{(k)}) = \text{tansig}(\zeta^{(k)}) \quad (4.8)$$

$\zeta^{(k)}$, $b_2^{(k)}$ and $\gamma^{(k)}$ are given by $(N_3 \times 1)$ column vectors and $w_2^{(k)}$ is a $(N_3 \times N_2)$ matrix. $w_2^{(k)}$ is the weight matrix in the second neuron layer and $b_2^{(k)}$ are the biases in the second neuron layer. The formula (2.19) corresponds to (4.1), where $w_2^{(k)} = W$. Both formulas defines the same matrix equation. The transfer function $f_2$ in the second neuron layer is given by:

$$f_2(x) = f_1(x) \quad (4.9)$$

Where $f_1(x)$ already has been defined in (4.3).

Neuron layer 3:

The inputs to this neuron layer is given by $\gamma^{(k)}$, which are the outputs from neuron layer 2. The following formulas are obtained for neuron layer 3:

$$\beta^{(k)} = w_3^{(k)} \gamma^{(k)} + b_3^{(k)} \quad (4.10)$$

$$u(k) = O^{(k)} = f_3(\beta^{(k)}) = \text{purelin}(\beta^{(k)}) = \beta^{(k)} \quad (4.11)$$

$\beta^{(k)}$, $b_3^{(k)}$ and $u(k) = O^{(k)}$ are given by $(N_4 \times 1)$ column vectors, but since $N_4 = 1$ they are all scalars. $w_3^{(k)}$ is a $(N_4 \times N_3)$ matrix, which becomes a $(1 \times N_3)$ row vector. $w_3^{(k)}$ is the weight "matrix" in the third neuron layer and $b_3^{(k)}$ is the bias in the third neuron layer. The output from this neuron layer, which is the output from the neural network is given by scalar $u(k) = O^{(k)}$. The control signal $u(k)$ is then used to calculate the error $E_1(k + 1)$. These calculations are studied in the next section. The transfer function $f_3$ in the third neuron layer is given by:
\[ f_3(x) = \text{purelin}(x) = x \]  

(4.12)

Which is a pure linear function with the gradient one, which passes through the origin.

Note that all controllers in this chapter have \( N_2 = 25 \), \( N_3 = 30 \) and \( N_4 = 1 \), when they are used in the simulations.

### 4.2 Calculations of the neural network error, \( E_I(k+1) \)

Figure 4.2.1. The neural network, \( NN^{(k)} \), is first used to test how well it can approximate a non-linear function. The mixed extremum controller is chosen as the test function. The training is done as shown in the picture. The same input signal, \( \xi^{(k)} = w(k) \), is used for the mixed extremum controller and the example controller in this section. The outputs from the controllers, \( u_{NN}(k) \) and \( u_{\text{mixed}}(k) \), see formula (3.25), are compared and the error \( E_I(k+1) \) can then be calculated. This error is used to adjust the example controller towards the mixed extremum controller.
The error function or cost function, $E_1(k+1)$, is introduced. The error, $E_1(k+1)$, is backpropagated through the neural network and the weights and bias are changed, so that this error is minimised. The method for obtaining the cost function, $E_1(k+1)$, is very similar with what is done in section 2.4. The difference is that a three neuron layer network is used. A simpler network, in which the output $O^{(i)} = u(k)$ are directly used to calculate the error $E_1(k+1)$, is studied first, see figure 4.2.1. The control signal $u(k)$ is then directly trained towards the mixed extremum controller, which have the control signal $u_{\text{mixed}}(k)$. $u_{\text{mixed}}(k)$ is given by $u(k)$ in formula (3.25). This example with a simple neural network is used, to test the neural network. The example controller that is related to the error $E_1(k+1)$ has the scalar input $z(k) = w(k)$, see formula (3.25). The following error function is introduced:

$$E_1(k+1) = \frac{1}{2} (u(k) - u_{\text{mixed}}(k))^2$$  \hspace{1cm} (4.13)

The variable $x^{(k)}$, which is any of the weights $w_i^{(k)}$ or any of the bias $b_i^{(k)}$ for $i = 1, 2$ or 3, is introduced. The following formulas are then obtained for the changes in weights and bias during step $k$:

$$x^{(k)} = x^{(k-1)} + \Delta x^{(k)} + \alpha \Delta x^{(k-1)}$$  \hspace{1cm} (4.14)

$$\Delta x^{(k)} = -\eta(k) \frac{\partial E_1(k+1)}{\partial x^{(k)}}$$  \hspace{1cm} (4.15)

$\alpha$ is the momentum parameter and $\eta(k)$ is the adaptive step length during step $k$, see formula (2.29)-(2.33) and (2.34)-(2.38) in section 2.4 and 2.5. $x^{(k)}$ is the weight or bias during step $k$ and $x^{(k-1)}$ is the weight or bias during step $k - 1$. $\Delta x^{(k)}$ is the change in the weight or bias during step $k$ and $\Delta x^{(k-1)}$ is the change in the weight or bias during step $k - 1$, this term is the momentum term in the neural network. Note that formula (4.14) and (4.15) contain twelve formulas. It comes from the fact that $x^{(k)}$ is any of the weights $w_i^{(k)}$ or any of the bias $b_i^{(k)}$ for $i = 1, 2$ or 3. Formula (4.15) states that the error $E_1(k+1)$ is backpropagated by the derivatives of this function with respect to the weight or bias $x^{(k)}$. Note that $u_{\text{mixed}}(k)$ doesn’t contain any weights or bias and is therefore considered to be a constant, when $E_1(k+1)$ is differentiated with respect to $x^{(k)}$. Using (4.13) in (4.15) and the chain rule the following formula is obtained:
\[\Delta x^{(k)} = -\eta(k) \frac{\partial E_1^{(k)}(k+1)}{\partial x^{(k)}} = -\eta(k) \frac{\partial}{\partial x^{(k)}} \left[ \frac{1}{2} (u^{(k)} - u_{\text{mixed}}^{(k)})^2 \right] = \]

\[= \eta(k) (u_{\text{mixed}}^{(k)} - u^{(k)}) \frac{\partial u^{(k)}}{\partial x^{(k)}} \] (4.16)

Note that \( \eta(k), u_{\text{mixed}}^{(k)}, \) and \( u^{(k)} \) are all scalars. The term \( \frac{\partial u^{(k)}}{\partial x^{(k)}} \) is given by the scalar \( u(k) \), which is differentiated with respect to the matrix or vector \( x^{(k)} \). This derivative is studied more closely in the following analysis. Before this derivative can be calculated it’s necessary to use the chain rule on matrices or vectors, that are differentiated, through the different neuron layers. Using the chain rule in matrix formulation on \( \frac{\partial u^{(k)}}{\partial \theta_i^{(k)}} \) for \( i = 1, 2 \) and 3 and noting that a row vector is obtained, but a column vector is desired, the following formulas are obtained:

\[\frac{\partial u^{(k)}}{\partial \theta_1^{(k)}} = \left( \frac{\partial u^{(k)}}{\partial \beta_1^{(k)}} \frac{\partial \beta_1^{(k)}}{\partial \gamma_1^{(k)}} \frac{\partial \gamma_1^{(k)}}{\partial \xi_1^{(k)}} \frac{\partial \xi_1^{(k)}}{\partial H_1^{(k)}} \right)^T \] (4.17)

\[\frac{\partial u^{(k)}}{\partial \theta_2^{(k)}} = \left( \frac{\partial u^{(k)}}{\partial \beta_2^{(k)}} \frac{\partial \beta_2^{(k)}}{\partial \gamma_2^{(k)}} \frac{\partial \gamma_2^{(k)}}{\partial \xi_2^{(k)}} \frac{\partial \xi_2^{(k)}}{\partial H_2^{(k)}} \right)^T \] (4.18)

\[\frac{\partial u^{(k)}}{\partial \theta_3^{(k)}} = 1 \] (4.19)

Formula (4.19) is obtained from (4.10) and (4.11). The \(^T\) stands for the transpose of the preceding matrix, vector or scalar. That the scalar \( \frac{\partial u^{(k)}}{\partial \theta_3^{(k)}} = 1 \) is easily seen from (4.11).

\( \frac{\partial u^{(k)}}{\partial \theta_1^{(k)}} \) and \( \frac{\partial u^{(k)}}{\partial \theta_2^{(k)}} \) are given by a \((1 \times N_2)\) column vector and \((1 \times N_3)\) column vector respectively, which is the way these derivatives are defined in the neural networks. By using the chain rule on a scalar valued function, that’s differentiated with respect to a vector, a row vector is obtained, see Persson and Böiers (88) [3] p. 117-119. This fact can be seen in \( \frac{\partial \beta^{(k)}}{\partial \gamma^{(k)}} \), which is a \((1 \times N_3)\) row vector. The remaining terms in (4.17) and (4.18) are matrices of different sizes. Note that on the right-hand side of the formulas given by (4.17)-(4.18) the convention from the chain rule, see Persson and Böiers (88) [3] p. 117-119, is used on the terms inside the brackets. The left-hand side of (4.17) or (4.18) is given by:
\[
\frac{\partial \mu(k)}{\partial \gamma_j^{(k)}} = \left( \begin{array}{c}
\frac{\partial \mu(k)}{\partial \gamma_j^{(k)}(1)} \\
\frac{\partial \mu(k)}{\partial \gamma_j^{(k)}(2)} \\
\vdots \\
\frac{\partial \mu(k)}{\partial \gamma_j^{(k)}(N_{i+1})}
\end{array} \right)
\]  
(4.20)

Note that a column vector is used for the bias in the neural networks, see formula (4.1) and (4.7), and the changes of the bias must therefore also be column vectors in accordance with (4.20). Formula (4.20) with \( i = 1 \) has \( N_2 \) columns and is related to (4.17). Similarly formula (4.20) with \( i = 2 \) has \( N_3 \) columns and is related to (4.18). Note that \( b_j^{(k)}(j) \) is element \( j \) in the column vector \( b_j^{(k)} \). A scalar is differentiated with respect to a column vector in (4.20) just as in \( \frac{\partial \gamma_j^{(k)}}{\partial \gamma_j^{(k)}} \), which is given by the following formula:

\[
\frac{\partial \gamma_j^{(k)}}{\partial \gamma_j^{(k)}} = \left( \begin{array}{ccc}
\frac{\partial \gamma_j^{(k)}}{\partial \gamma_1^{(k)}} & \frac{\partial \gamma_j^{(k)}}{\partial \gamma_2^{(k)}} & \cdots & \frac{\partial \gamma_j^{(k)}}{\partial \gamma_{N_3}^{(k)}}
\end{array} \right) = w_j^{(k)}
\]  
(4.21)

This is a \((1 \times N_3)\) row vector and not a column vector as in (4.20). The last equality is obtained from formula (4.10). Note that \( \gamma_j^{(k)} \) is element \( j \) in the column vector \( \gamma^{(k)} \). Before the chain rule can be applied on the formulas (4.7) and (4.8) in neuron layer 2, a new operator on matrices needs to be introduced. \( \text{diag}(h) \) of a row vector or column vector, with \( N \) rows or columns respectively, is a \((N \times N)\) matrix, where the elements in the diagonal are the only non zero elements. The column or row vector is placed in the diagonal. For instance, if \( h \) is a \((3 \times 1)\) column vector, \( \text{diag}(h) \) is defined in the following way:

\[
\text{diag}(h) = \begin{pmatrix}
h_1 & 0 & 0 \\
0 & h_2 & 0 \\
0 & 0 & h_3
\end{pmatrix}
\]  
(4.22)

Where \( h_j \) is element \( j \) in the column vector \( h \). Note that \( \text{diag}(h) = h \) in the special case, when \( h \) is given by a scalar. Using formulas (4.7) and (4.8) in neuron layer 2 and the chain rule in matrix formulation, see Persson and Böiers (88) [3] p. 117-119, the following formulas are obtained:

\[
\frac{\partial \gamma_j^{(k)}}{\partial \zeta_j^{(k)}} = \text{diag}(f'_2(\zeta_j^{(k)}))
\]  
(4.23)
\[
\frac{\partial \gamma^{(k)}_j}{\partial \mathbf{w}_j^{(k)}} = w^{(k)}_j
\]  
(4.24)

\[
\frac{\partial \gamma^{(k)}_j}{\partial \mathbf{b}_j^{(k)}} = \text{diag}(f'_2(\zeta^{(k)}))
\]  
(4.25)

Since element \( j \) in the column vector \( \gamma^{(k)} \) is given by \( \gamma^{(k)}_j = f_2(\zeta^{(k)}_j) \) all derivatives of the type \( \frac{\partial \gamma^{(k)}_j}{\partial \zeta^{(k)}_i} \), where \( i \neq j \), are zero. From this fact formula (4.23) is obtained. Similarly \( \gamma^{(k)}_j \) is a function of \( b^{(k)}_j \) and doesn’t depend on \( b^{(k)}_i \), with \( i \neq j \). This implies that the derivatives of the type \( \frac{\partial \gamma^{(k)}_j}{\partial b^{(k)}_i} \), with \( i \neq j \), are zero and formula (4.25) is obtained from this fact. That \( \frac{\partial \gamma^{(k)}_j}{\partial h^{(k)}_i} = w^{(k)}_j(i,j) \) can easily be seen from (2.19), where \( w^{(k)}_j(i,j) \) is the element in row \( i \) and column \( j \) in the matrix \( w^{(k)}_j \). From this fact formula (4.24) is obtained. The derivative \( f'_2 \) is given by formula (2.4). Using formulas (4.1) and (4.2) in neuron layer 1 and the chain rule in matrix formulation, the following formula is obtained:

\[
\frac{\partial \mathbf{H}^{(k)}_i}{\partial \mathbf{b}_i^{(k)}} = \text{diag}(f'_1(h^{(k)}))
\]  
(4.26)

Where \( H^{(k)}_j \) is a function of \( b^{(k)}_j(j) \) and does not depend on \( b^{(k)}_i(i) \), with \( i \neq j \). This fact implies that the derivatives of the type \( \frac{\partial \mathbf{H}^{(k)}_j}{\partial \mathbf{b}_i^{(k)}(i)} \) are zero, for \( i \neq j \). From this fact formula (4.26) is obtained. The derivative \( f'_1 \) is given by formula (2.4). All the terms in (4.17)-(4.19) have been found and formula (4.17) and (4.18) can now be expressed as:

\[
\frac{\partial \mathbf{d}^{(k)}}{\partial \mathbf{b}_i^{(k)}} = \left[w^{(k)}_i \text{diag}(f'_2(\zeta^{(k)}))w^{(k)}_i \text{diag}(f'_1(h^{(k)}))\right]^T
\]  
(4.27)

\[
\frac{\partial \mathbf{d}^{(k)}}{\partial \mathbf{b}_j^{(k)}} = \left[w^{(k)}_j \text{diag}(f'_2(\zeta^{(k)}))\right]^T
\]  
(4.28)

Where the formulas (4.21) and (4.23)-(4.26) have been used.

33
The matrix, that contains the changes in weights, can be derived by using the chain-rule and some functions in Matlab. One matrix with the changes in weights is needed for each neuron layer in the neural network, which means that three matrices are needed. Using the chain-rule in matrix formulation, see Persson and Böiers (88) [3] p. 117-119, and remembering that the result is a row vector, but a matrix, with the changes in weights, is desired in the neural network, the following results are obtained:

\[
\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_1^{(k)}} = \frac{\partial \mathbf{d}(k)}{\partial \mathbf{b}^{(k)}} \frac{\partial \mathbf{b}^{(k)}}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{a}^{(k)}} \frac{\partial \mathbf{a}^{(k)}}{\partial \mathbf{w}_1^{(k)}} = \\
= \frac{\partial \mathbf{d}(k)}{\partial \mathbf{b}^{(k)}} \frac{\partial \mathbf{b}^{(k)}}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{a}^{(k)}}\text{kron}(\text{eye}(N_2),([\mathbf{e}^{(k)}]^T) = \\
= \hat{d}^{(k)}\text{kron}(\text{eye}(N_2),([\mathbf{e}^{(k)}]^T)
\] 

(4.29)

\[
\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_1^{(k)}} = \left[\text{reshape}\left(\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_1^{(k)}},N_1,N_2\right)\right]^T
\] 

(4.30)

\[
\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_2^{(k)}} = \frac{\partial \mathbf{d}(k)}{\partial \mathbf{b}^{(k)}} \frac{\partial \mathbf{b}^{(k)}}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{a}^{(k)}}\text{kron}(\text{eye}(N_3),([\mathbf{h}^{(k)}]^T) = \\
= \hat{e}^{(k)}\text{kron}(\text{eye}(N_3),([\mathbf{h}^{(k)}]^T)
\] 

(4.31)

\[
\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_2^{(k)}} = \left[\text{reshape}\left(\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_2^{(k)}},N_2,N_3\right)\right]^T
\] 

(4.32)

\[
\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_3^{(k)}} = \frac{\partial \mathbf{d}(k)}{\partial \mathbf{b}^{(k)}} = \\
= \frac{\partial \mathbf{d}(k)}{\partial \mathbf{b}^{(k)}}\text{kron}(\text{eye}(N_4),([\mathbf{y}^{(k)}]^T) = \\
= \hat{f}^{(k)}\text{kron}(\text{eye}(N_4),([\mathbf{y}^{(k)}]^T)
\] 

(4.33)

\[
\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_3^{(k)}} = \left[\text{reshape}\left(\frac{\partial \mathbf{d}(k)}{\partial \mathbf{w}_3^{(k)}},N_3,N_4\right)\right]^T
\] 

(4.34)
Where the scalar \( \frac{\partial a(k)}{\partial \beta(k)^{(k)}} = 1 \). \( \hat{d}^{(k)} \), \( \hat{e}^{(k)} \) and \( \hat{f}^{(k)} \) are defined in the formulas (4.29), (4.31) and (4.33). \( \hat{d}^{(k)} \) is a \( (1 \times N_2) \) row vector, \( \hat{e}^{(k)} \) is a \( (1 \times N_3) \) row vector and \( \hat{f}^{(k)} \) is a scalar. \( \frac{\partial a(k)}{\partial \omega_1^{(k)}} \) is a \( (1 \times N_2, N_1) \) row vector and \( \frac{\partial a(k)}{\partial \omega_2^{(k)}} \) is a \( (1 \times N_3, N_2) \) row vector. That row vectors are obtained in (4.29) and (4.31) are in accordance with chain-rule in matrix formulation. 

\( \frac{\partial a(k)}{\partial \omega_1^{(k)}} \) is a \( (N_2 \times N_1) \) matrix, which have the same matrix size as \( w_1^{(k)} \) and \( \frac{\partial a(k)}{\partial \omega_2^{(k)}} \) is a \( (N_3 \times N_2) \) matrix, which have the same matrix size as \( w_2^{(k)} \), all in accordance with how these matrices are defined in the neural networks. Since \( N_4 = 1 \) the formula (4.34) isn’t necessary, because \( w_{1}^{(k)} \) already is a row vector and thus 

\[ \frac{\partial H^{(k)}}{\partial h^{(k)}} = \frac{\partial a(k)}{\partial \omega_1^{(k)}} = \frac{\partial a(k)}{\partial \omega_2^{(k)}}, \]

which both are \( (1 \times N_3) \) row vectors. Formula (4.34) transposes the right-hand side of (4.33) two times and the same row vector of size \( (1 \times N_3) \) is obtained. The matlab function \( \text{eye}(N_i) \), for \( i = 2, 3 \) and 4, gives a \( (N_i \times N_i) \) identity matrix and of course, for \( N_4 = 1 \), a scalar with the value one is obtained. How the matlab functions \( \text{kron} \) and \( \text{reshape} \) are used in the formulas (4.29)-(4.34), are explained later. Using the chain-rule in matrix formulation and formula (4.1) and (4.2), the following formula for \( \frac{\partial H^{(k)}}{\partial h^{(k)}} \) is obtained:

\[ \frac{\partial H^{(k)}}{\partial h^{(k)}} = \text{diag}\left( f_1'(h^{(k)}) \right) \]  

(4.35)

Where \( H_j^{(k)} \) is a function of \( h_j^{(k)} \) and does not depend on \( h_i^{(k)} \), with \( i \neq j \). This fact implies that the derivatives of the type \( \frac{\partial H_j^{(k)}}{\partial h_i^{(k)}} \) are zero, for \( i \neq j \). From this fact formula (4.35) is obtained. The formula (4.35) can be compared with (4.26), where a similar result has been obtained.

With the exception of the matrices that contain \( \text{kron} \) and \( \text{reshape} \) all the terms in formula (4.29)-(4.34) have now been explained in this section. A simple example is given to explain formula (4.31) and (4.32). It evident that formula (4.29)-(4.30), (4.31)-(4.32) and (4.33)-(4.34) are obtained in a very similar way and that it’s only necessary to study for instance (4.31)-(4.32). To make the calculations simpler to follow, it’s assumed that \( N_2 = 2 \) and \( N_3 = 3 \). Once the results from this simple example is obtained, it’s not difficult to generalise to arbitrary values of \( N_2 \) and \( N_3 \). Formula (4.31) and (4.32) with \( N_2 = 2 \) and \( N_3 = 3 \) become:
\[ \dot{\alpha}_d(k) = \varepsilon^{(k)} \kron \left( \text{eye} (N_3), (H^{(k)})^T \right) = \]

\[ = \begin{pmatrix} \dot{\varepsilon}_1^{(k)} & \dot{\varepsilon}_2^{(k)} & \dot{\varepsilon}_3^{(k)} \end{pmatrix} \begin{pmatrix} (H^{(k)})^T & 0 & 0 \\ 0 & (H^{(k)})^T & 0 \\ 0 & 0 & (H^{(k)})^T \end{pmatrix} = \]

\[ = \begin{pmatrix} \dot{\varepsilon}_1^{(k)} & \dot{\varepsilon}_2^{(k)} & \dot{\varepsilon}_3^{(k)} \end{pmatrix} \begin{pmatrix} H_1^{(k)} & H_2^{(k)} & 0 & 0 & 0 & 0 \\ 0 & 0 & H_1^{(k)} & H_2^{(k)} & 0 & 0 \\ 0 & 0 & 0 & 0 & H_1^{(k)} & H_2^{(k)} \end{pmatrix} = \]

\[ = \begin{pmatrix} \dot{\varepsilon}_1^{(k)} H_1^{(k)} & \dot{\varepsilon}_1^{(k)} H_2^{(k)} & \dot{\varepsilon}_2^{(k)} H_1^{(k)} & \dot{\varepsilon}_2^{(k)} H_2^{(k)} & \dot{\varepsilon}_3^{(k)} H_1^{(k)} & \dot{\varepsilon}_3^{(k)} H_2^{(k)} \end{pmatrix} \quad \text{(4.36)} \]

\[ \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}} = \left[ \text{reshape} \left( \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}}, N_2, N_3 \right) \right]^T = \left[ \begin{pmatrix} \dot{\varepsilon}_1^{(k)} H_1^{(k)} & \dot{\varepsilon}_1^{(k)} H_2^{(k)} \\ \dot{\varepsilon}_2^{(k)} H_1^{(k)} & \dot{\varepsilon}_2^{(k)} H_2^{(k)} \\ \dot{\varepsilon}_3^{(k)} H_1^{(k)} & \dot{\varepsilon}_3^{(k)} H_2^{(k)} \end{pmatrix} \right]^T = \]

\[ = \begin{pmatrix} \dot{\varepsilon}_1^{(k)} H_1^{(k)} & \dot{\varepsilon}_1^{(k)} H_2^{(k)} \\ \dot{\varepsilon}_2^{(k)} H_1^{(k)} & \dot{\varepsilon}_2^{(k)} H_2^{(k)} \\ \dot{\varepsilon}_3^{(k)} H_1^{(k)} & \dot{\varepsilon}_3^{(k)} H_2^{(k)} \end{pmatrix} = \begin{pmatrix} \varepsilon^{(k)} \end{pmatrix}^T \begin{pmatrix} H^{(k)} \end{pmatrix}^T \quad \text{(4.37)} \]

How \( \kron \left( \text{eye} (N_3), (H^{(k)})^T \right) \) is used in the neural networks can be seen in formula (4.36). \((H^{(k)})^T\) is of course a \((1 \times N_2)\) row vector and \(\dot{\varepsilon}_i^{(k)}\) for \(i = 1, 2\) and 3 is a scalar.

It’s desirable to obtain a scalar valued function, \( \alpha(k) \), that is differentiated with respect to a matrix, \( w_2^{(k)} \). A trick is used to obtain the \((N_3 \times N_2)\) matrix, \( \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}} \). Instead of obtaining \( \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}} \) directly, \( \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}} \) is obtained first. The chain-rule in matrix formulation, see Persson and Böiers (88) [3] p. 117-119, is used to obtain \( \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}} \), where a scalar valued function is differentiated with respect to a long row vector, \( \bar{w}_2^{(k)} \). By using \( \kron \left( \text{eye} (N_3), (H^{(k)})^T \right) \) and the chain-rule in matrix formulation, the rows of \( \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}} \) are put after each other to create the row vector \( \frac{\partial \alpha_d(k)}{\partial w_2^{(k)}} \), see (4.36) and (4.37). Note that \( \kron \left( \text{eye} (N_3), (H^{(k)})^T \right) \) is given by a
\( (N_2 \times N_3 N_2) \) matrix, see (4.36). By using \( \text{reshape} \left( \frac{\partial \mathbf{d}(k)}{\partial W_2^{(k)}}, N_2, N_3 \right) \) the row vector \( \frac{\partial \mathbf{d}(k)}{\partial W_2^{(k)}} \) is divided into \( N_3 \) rows of the size \( (1 \times N_2) \) and then these rows are used as columns, to create the \( (N_2 \times N_3) \) matrix \( \left( \frac{\partial \mathbf{d}(k)}{\partial W_2^{(k)}} \right)^T \). Once this result is transposed \( \frac{\partial \mathbf{d}(k)}{\partial W_2^{(k)}} \) is obtained, see formula (4.37). The results, which are obtained from this example, illustrate how the calculations in the formulas (4.29)-(4.34) are made.

That \( \frac{\partial \mathbf{d}(k)}{\partial W_2^{(k)}}(i,j) = \hat{e}_i H_j^{(k)} \) can be seen from (4.37). The result from (4.37) can be compared with the results that are obtained from (2.13) and (2.14) in section 2.3. The fact that \( \frac{\partial E(t+1)}{\partial W_2^{(k)}} = k_1 \cdot \frac{\partial \mathbf{d}(k)}{\partial W_2^{(k)}} \), where \( k_1 \) is a scalar constant, can be seen from (4.16). Note that the corresponding scalar constant \( k_1 \) is given by \( \eta \left( C^{(\mu)}_i - O^{(\mu)}_j \right) \) in (2.13) and (2.14). It must be a scalar, if the two neural networks are to be compared. By noting that \( \xi_j^{(k)} \) in (2.14) corresponds to \( H_j^{(k)} \) in (4.37), it can be seen that result in (2.14) has the same structure as the one given by (4.37).

For all controllers in this chapter, neuron layer 3 is not updated by the backpropagation algorithm. In other words, formula (4.33) and (4.34) are not used and formula (4.14)-(4.16) are not used, when \( i = 3 \). This means that \( b_3^{(k)} \) and \( w_3^{(k)} \) are constants matrices, that don’t change their values, during the training of the neural networks in this chapter. The reason, why formula (4.33)-(4.34) and (4.14)-(4.16), with \( i = 3 \), are shown, is to illustrate how neural networks are implemented. Since the third neuron layer is not updated, the neural networks in this chapter are networks with only two active neuron layers. Neuron layer 3 is only used to calculate \( u(k) \).

To only use two active neuron layers is a choice that is made. The third neuron layer (the output layer), which only contains one neuron, can also be updated, but it’s then necessary to use different step lengths, for each neuron layer. The step length \( \eta \) for a weight is divided by the number of input signals to the neuron, that corresponds to the weight in question, see Hassoun (95) [10] p. 206 and 211-212. Although the author don’t mention, how the step length ought to be chosen for the bias, I think it’s possible to apply the same type of rules for choosing the step length \( \eta \) for the bias, as is done for the weights. By scaling the weights (and bias) in this manner, the neural networks are working better and the convergence times are decreasing. For the three controllers, that are based on neural networks, the step lengths for the weights and the bias ought to be scaled in the following way (if all three neuron layers are to be active): the weights and bias in the first neuron layer are divided by \( (1 + N_1) \), in the second by \( (1 + N_2) \) and in the third by \( (1 + N_3) \). Note that \( N_1 = 1 \) (for controller 1) or 5 (for controller 2 and controller 3) is the number of input signals to the neural network, \( N_2 = 25 \) is the number of neurons in the second neuron layer and \( N_3 = 30 \) is the number of neurons in the third neuron layer.
Graph 4.2.1. The result of a simulation of the controller in figure 4.2.1 is shown. The output $u(k)$ for the example controller based on neural networks, the lower dashed curve, and the output from the mixed extremum controller $u_{\text{mixed}}(k)$, the upper curve, are plotted as a function of the input $w(k)$. The function $w(k)$ increases linearly from $-3$ to $3$ with the step length $0.05$ during the plot.

The control signal $u(k)$ for the example controller as a function of $w(k)$ can be seen in graph 4.2.1. Note that $u_{\text{mixed}}(k)$, the output from the mixed extremum controller during step $k$, is given by formula (3.25), where the plus sign is chosen.

During the training of the example controller a random input signal $w(k)$ is used, that has a standard deviation of $2$ and a mean value of zero. The size of the neural network in the example controller is given by $N_1 = N_4 = 1$, $N_2 = 25$ and $N_3 = 30$. The momentum parameter $\alpha$ in formula (4.14) has the value $0.1$ and no adaptive step length is used, in other words $a = d = 0$ in (2.29). The neural network is trained with $100\,080$ steps and is then simulated during $5000$ steps with the standard deviation $\sigma_v = 0.2$. Once the neural network in the example controller has been trained, the weights $w_i^{(k)}$ and the bias $b_i^{(k)}$ for $i = 1, 2$ and $3$ and $k = 100\,080$, see formula (4.1)-(4.10), are saved and used in the simulations of the example controller.
Graph 4.2.2. The losses for the example controller, that is trained 100 080 steps, are given by the solid curve and the losses for the mixed extremum controller are given by the dashed dotted curve. The losses for the constant extremum controller, which have the largest losses, are given by the dotted curve. 
\( \sigma_e = 0.2 \) in the simulation.

Two different values on the step length \( \eta \) are used during the training of the neural network: \( \eta_{\text{start}} \) and \( \eta_{\text{fix}} \), see table 4.2.1 and formula (4.15). The result from the graph 4.2.1 is that the output \( u(k) \) for the example controller goes towards the output \( u_{\text{mixed}}(k) \) for the mixed extremum controller, once the neural network has been trained. The control law \( u(k) \) for the example controller don’t follow \( u_{\text{mixed}}(k) \) too well, when \( w(k) \) lies close to 0.125, since a peak appears in \( u_{\text{mixed}}(k) \) at \( w(k) = 0.125 \), see formula (3.25). This discontinuity in the output \( u(k) \) affects the weights and the bias. The weights and the bias are not converging too well, see graph 4.2.3 and graph 4.2.4.
Graph 4.2.3. The weights are plotted during 100 080 steps of training for the example controller. The changes of a diagonal and an off diagonal element of the weight matrices in neuron layer one and two can be seen. The two top graphs shows the changes of two elements of the weight matrix in neuron layer one and the two bottom graphs shows similar elements in neuron layer two, see formula (4.1) and formula (4.7).

The losses for the example controller, when $\sigma_e = 0.2$ during the simulations, are compared with the corresponding losses for the mixed extremum controller and the constant extremum controller, see graph 4.2.2. The losses for the controllers in graph 4.2.2 are calculated just as the losses are calculated for the controllers in the next chapter, see table 4.2.1 and 5.6.1. For the example controller the losses are 494,1331, when $\sigma_e = 0.2$. The constant extremum controller has larger losses than the example controller and the mixed extremum controller has lower losses than the example controller, when $\sigma_e = 0.2$, see table 5.6.1. Note that the simulation of the example controller is done in exactly the same way as the simulation of any of the controllers in chapter 5, when $\sigma_e = 0.2$.

<table>
<thead>
<tr>
<th>The number of steps of training:</th>
<th>The step length, $\eta$:</th>
<th>The momentum parameter, $\alpha$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1-240</td>
<td>$\eta_{start} = 0.005$</td>
<td>$\alpha = 0.1$</td>
</tr>
<tr>
<td>Step 241-100 080</td>
<td>$\eta_{fix} = 0.05$</td>
<td>$\alpha = 0.1$</td>
</tr>
</tbody>
</table>

Table 4.2.1. Important variables and parameters that are used during the training of the neural network in the example controller can be seen.
Graph 4.2.4. The biases are plotted during 100,080 steps of training for the example controller. The changes of the two first elements of the bias vector in neuron layer one and two can be seen. The two top graphs show the changes of two elements of the bias vector in neuron layer one and the two bottom graphs show similar elements in neuron layer two, see formula (4.1) and formula (4.7).
4.3 Controller 1 based on neural networks

The input, $\xi^{(k)}$, to controller 1 during step $k$ is given by the scalar $w(k-1)$ and the output is given by the scalar $u(k)$, see figure 4.3.1. Since the error $E(k+1)$ contains $w(k)$, the network is a kind of recurrent network. This means that a signal in the last "output", the error $E(k+1)$, is later used as an input. The error $E(k+1)$ that is backpropagated through the neural network is given by:

$$E(k+1) = p^2(k+1) = (\hat{y}(k+1) - y_0)^2$$

(4.38)

Where $p(k+1)$ is given by formula (3.16). Comparing formula (4.38) and (4.13), it can be seen that controller 1 in this section isn't based on a "pure" neural network model with backpropagation, as in the example controller in section 4.2. The difference between the two controllers, one given in this section and the other given in the preceding section, is that the error $E(k+1)$ in (4.38) is based on $\hat{y}(k+1)$ and the error $E_1(k+1)$ is based on $u(k)$.

Using (3.17) and (3.18), $w(k)$ and $p(k+1)$ can be expressed as:

$$w(k) = -c_1w(k-1) + b_0 + (b_{11} - c_1b_{10})u(k-1) + (-c_1b_{20})u^2(k-1) +$$

$$+ (c_1 - a_1)y(k) - (1 + c_1)y_0$$

(4.39)

$$p(k+1) = b_{10}u(k) + b_{20}u^2(k) + w(k)$$

(4.40)

The reason why these formulas are rewritten here is that it shows how this formulas are implemented in Matlab and that it makes the following analysis easier to follow, when $p(k+1)$ is differentiated with respect to $u(k)$ and $u(k-1)$. The formulas (4.14) and (4.15) hold also for this controller, but $\frac{\partial E_1(k+1)}{\partial \xi^{(k)}}$ must be changed to $\frac{\partial E(k+1)}{\partial \xi^{(k)}}$ with $E(k+1)$. 

42
given by (4.38). \( E(k+1) \) is differentiated with respect to \( u(k) \) and \( u(k-1) \), since \( y(k+1) \) and therefore \( \dot{y}(k+1) \) is given by a dynamical process containing \( u(k) \) and \( u(k-1) \). Using the "chain-rule" the following formula is obtained for \( \frac{\partial E(k+1)}{\partial x^{(k)}} \):

\[
\frac{\partial E(k+1)}{\partial x^{(k)}} = 2p(k+1) \left[ \frac{\partial \hat{y}(k+1)}{\partial \hat{u}(k)} \frac{\partial \hat{u}(k)}{\partial x^{(k)}} + \frac{\partial \hat{y}(k+1)}{\partial \hat{u}(k-1)} \frac{\partial \hat{u}(k-1)}{\partial x^{(k-1)}} \right] =
\]

\[
= 2p(k+1) \left[ \frac{\partial \hat{y}(k+1)}{\partial \hat{u}(k)} \frac{\partial \hat{u}(k)}{\partial x^{(k)}} + \frac{\partial \hat{y}(k+1)}{\partial \hat{u}(k-1)} \frac{\partial \hat{u}(k-1)}{\partial x^{(k-1)}} \right] =
\]

\[
= 2p(k+1) \left[ (2b_{20}u(k) + b_{10}) \frac{\partial \hat{u}(k)}{\partial x^{(k)}} + b_{11} \frac{\partial \hat{u}(k-1)}{\partial x^{(k-1)}} \right]
\]

(4.41)

Note that some terms are neglected, when the chain-rule is used, to insure that the right solution is obtained, when the "constant extremum controller" is used in the controllers based on neural networks. The control law that is desired, when the "constant extremum controller" is needed in the controllers based on neural networks, is given by \( u(k) = -0.25 \), see formula (3.25). \( x^{(k)} \) is any of the weight matrices \( w_i^{(k)} \) or bias vectors \( b_i^{(k)} \) for \( i = 1 \) or 2. \( \frac{\partial \hat{u}(k-1)}{\partial x^{(k-1)}} \) is obtained from (4.17)-(4.18) and (4.29)-(4.32) with the superscript \( k \) changed to \( k-1 \), which means that all matrices, vectors and scalars are taken from step \( k-1 \). Note that this means that the neural network has some kind of memory. \( \frac{\partial \hat{u}(k)}{\partial x^{(k)}} \) is obtained directly from (4.17)-(4.18) and (4.29)-(4.32). From (3.17) it can be seen that \( p(k+1) \) only is zero, when the minimum variance controller can be used in the mixed extremum controller, see formula (3.25). \( \frac{\partial E(k+1)}{\partial x^{(k)}} \rightarrow 0 \), when the neural network converges towards a solution. The mixed extremum controller also contains a solution based on the constant extremum controller, see the formulas (3.7)-(3.9). This solution is obtained by setting the bracket in (4.41) to zero. It has been empirically found that \( \frac{\partial \hat{u}(k)}{\partial x^{(k)}} \rightarrow \frac{\partial \hat{u}(k-1)}{\partial x^{(k-1)}} \) and that \( u(k) \rightarrow u(k-1) \rightarrow u_0 \), when the controller based on neural networks operates as a constant extremum controller. Setting the bracket in (4.41) to zero and solving for \( u_0 \), the following expression is obtained for \( u_0 \):

\[
u_0 = - \frac{(b_{10} + b_{11})}{2b_{20}} = - \frac{B_1 * (1)}{2B_2 * (1)} = -0.25
\]

43
4.4 Controller 2 based on neural networks

\[ j(k-1) \quad u(k-2) \quad u(k-1) \quad u^2(k-1) \quad y(k) \]

The input vector, \( \xi^{(k)} \)

\[ \xrightarrow{NN^{(k)}} u(k) \quad \ldots \quad E(k+1) \]

The error:

This controller is very similar with controller 1 in the preceding section. Except formula (4.39), (4.40) and the input \( \xi^{(k)} \), all formulas and results for controller 1 in the preceding section also apply for controller 2 in this section. Instead of using formula (4.39) and (4.40) in the controller, the following formula is used:

\[ \hat{y}(k+1) = -c_1 j(k) + b_0 + b_{10} u(k) + b_{11} u(k-1) + b_{20} u^2(k) + (c_1 - a_1) y(k) \quad (4.42) \]

Where formula (3.13) has been used. It has been rewritten here, to show how this formula is implemented in Matlab for controller 2. The input, \( \xi^{(k)} \), during step k is a \( (1 \times 5) \) column vector containing the elements \( j(k-1) \), \( u(k-2) \), \( u(k-1) \), \( u^2(k-1) \) and \( y(k) \), see figure 4.4.1. Note that the controller 2 is a kind of recurrent network, since it is based on the error \( E(k+1) \), given by (4.39). Taking the differences, that has been mentioned in this section in account, the formulas, results and explanations for controller 1 can also be used for controller 2. When the formulas (4.17)-(4.19) an (4.29)-(4.34) are used, they contain different matrix multiplications, since \( N_1 = 1 \) for controller 1 and \( N_1 = 5 \) for controller 2. This fact and formula (4.42) imply that formula (4.41) are different for controller 1 and controller 2, even though the formula is written in the same way in both controllers.
4.5 Controller 3 based on neural networks

This controller is very similar to controller 2 in the preceding section. The output from controller 3 in this section is given by the scalar \( u(k) \). The same input vectors, \( \xi^{(k)} \), is used for controller 2 and for controller 3 in this section. The formula (4.42) is used instead of the formulas (4.39) and (4.40) in controller 3 in this section. All the results, formulas and explanations that are used for controller 2 with the exception of formula (4.41), can also be used for controller 3. The difference between controller 2 and controller 3 is that \( E(k + 1) \) in controller 3 is differentiated with respect to \( u(k) \), \( u(k - 1) \) and \( u(k - 2) \) and not only with respect to \( u(k) \) and \( u(k - 1) \) as is done in controller 2. Using the "chain-rule" and the fact that \( \frac{\partial p(k+1)}{\partial f(k+1)} = 1 \), the following formula is obtained for \( \frac{\partial E(k+1)}{\partial \xi^{(k)}} \):

\[
\frac{\partial E(k+1)}{\partial \xi^{(k)}} = 2 p(k+1) \left[ \frac{\partial \xi(k+1)}{\partial u(k)} \frac{\partial u(k)}{\partial \xi^{(k)}} + \frac{\partial \xi(k+1)}{\partial u(k-1)} \frac{\partial u(k-1)}{\partial \xi^{(k)}} + \frac{\partial \xi(k+1)}{\partial u(k-2)} \frac{\partial u(k-2)}{\partial \xi^{(k-2)}} \right] +
\]

\[
+ \frac{\partial \xi(k+1)}{\partial \xi(k)} \left( \frac{\partial \xi(k)}{\partial u(k-1)} \frac{\partial u(k-1)}{\partial \xi^{(k-1)}} + \frac{\partial \xi(k)}{\partial u(k-2)} \frac{\partial u(k-2)}{\partial \xi^{(k-2)}} \right) +
\]

\[
+ \frac{\partial \xi(k+1)}{\partial \xi(k)} \left( \frac{\partial \xi(k)}{\partial u(k-1)} \frac{\partial u(k-1)}{\partial \xi^{(k-1)}} + \frac{\partial \xi(k)}{\partial u(k-2)} \frac{\partial u(k-2)}{\partial \xi^{(k-2)}} \right) \right] =
\]

\[
= 2 p(k+1) \left[ \left( 2 b_{20} u(k) + b_{10} \right) \frac{\partial u(k)}{\partial \xi^{(k)}} + b_{11} \frac{\partial u(k-1)}{\partial \xi^{(k)}} + \right.
\]

\[
+ \left( c_1 - a_1 \left( 2 b_{20} u(k - 1) + b_{10} \right) \frac{\partial u(k - 1)}{\partial \xi^{(k-1)}} + b_{11} \frac{\partial u(k - 2)}{\partial \xi^{(k-2)}} \right) +
\]

\[
+ \left( - c_1 \left( 2 b_{20} u(k - 1) + b_{10} \right) \frac{\partial u(k - 1)}{\partial \xi^{(k-1)}} + b_{11} \frac{\partial u(k - 2)}{\partial \xi^{(k-2)}} \right) \right] \]  

(4.43)

The derivatives are chosen in exactly the same manner as it is done for controller 1, to obtain the solutions \( u(k) = -0.25 \) and \( p(k + 1) = 0 \). The explanations to formula (4.41) can also be used here (see also table 4.5.1).
<table>
<thead>
<tr>
<th>Matrices (for all $k$)</th>
<th>Size</th>
<th>Vectors (for all $k$)</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1^{(k)}$, $\frac{\partial E(k+1)}{\partial w_1^{(k)}}$, $\frac{\partial u(k)}{\partial w_1^{(k)}}$</td>
<td>$(N_2 \times N_1)$</td>
<td>$\xi^{(k)}$ (The input for controller 2 and controller 3 in this chapter. Note that $N_1 = 5$ for these controllers.)</td>
<td>$(N_1 \times 1)$</td>
</tr>
<tr>
<td>$w_2^{(k)}$, $\frac{\partial E(k+1)}{\partial w_2^{(k)}}$, $\frac{\partial u(k)}{\partial w_2^{(k)}}$, $\frac{\partial \xi^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(N_1 \times N_2)$</td>
<td>$h^{(k)}$, $H^{(k)}$, $b_1^{(k)}$, $\frac{\partial E(k+1)}{\partial b_1^{(k)}}$, $\frac{\partial u(k)}{\partial b_1^{(k)}}$</td>
<td>$(N_2 \times 1)$</td>
</tr>
<tr>
<td>$\frac{\partial \xi^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \gamma^{(k)}}{\partial b_2^{(k)}}$</td>
<td>$(N_1 \times N_3)$</td>
<td>$\xi^{(k)}$, $\gamma^{(k)}$, $b_2^{(k)}$, $\frac{\partial E(k+1)}{\partial b_2^{(k)}}$, $\frac{\partial u(k)}{\partial b_2^{(k)}}$</td>
<td>$(N_2 \times 1)$</td>
</tr>
<tr>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(N_2 \times N_2)$</td>
<td>$\tilde{d}^{(k)}$</td>
<td>$(N_1 \times 1)$</td>
</tr>
<tr>
<td>$rac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(N_2 \times N_2 N_1)$</td>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(1 \times N_3)$</td>
</tr>
<tr>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(N_2 \times N_2 N_1)$</td>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(N_2 \times N_1)$</td>
</tr>
<tr>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(N_2 \times N_2 N_1)$</td>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(1 \times N_2 N_1)$</td>
</tr>
<tr>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(N_2 \times N_2 N_1)$</td>
<td>$\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$, $\frac{\partial \theta^{(k)}}{\partial \theta^{(k)}}$</td>
<td>$(1 \times N_2 N_1)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scalars (for all $k$)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta^{(k)}$, $O^{(k)}$, $u(k)$, $b_3^{(k)}$, $u_{\text{mixed}}(k)$, $E_1(k+1)$, $E(k+1)$, $\eta(k)$, $\hat{j}^{(k)}$, $\hat{y}(k+1)$, $y(k+1)$, $p(k+1)$, $w(k)$, $u_0$, $\hat{u}_0$, $y_0$, $\hat{y}_0$, $\frac{\partial E(k+1)}{\partial \beta^{(k)}}$, $\frac{\partial u(k)}{\partial \beta^{(k)}}$, $\frac{\partial \eta(k)}{\partial \beta^{(k)}}$, $\frac{\partial \hat{j}^{(k)}}{\partial \beta^{(k)}}$, $\frac{\partial \hat{y}(k+1)}{\partial \beta^{(k)}}$, $\frac{\partial y(k+1)}{\partial \beta^{(k)}}$, $\frac{\partial p(k+1)}{\partial \beta^{(k)}}$, $\frac{\partial w(k)}{\partial \beta^{(k)}}$, $\frac{\partial u_0}{\partial \beta^{(k)}}$, $\frac{\partial \hat{u}_0}{\partial \beta^{(k)}}$, $\frac{\partial y_0}{\partial \beta^{(k)}}$, $\frac{\partial \hat{y}_0}{\partial \beta^{(k)}}$</td>
<td></td>
</tr>
</tbody>
</table>

$\xi^{(k)}$ (The scalar input for controller 1 and the example controller in section 4.2.)

Table 4.5.1. A summary of important matrices, vectors and scalars and their sizes are shown. Note that the matrices, vectors and scalars that are shown are in some way related to the calculations of the controllers based on neural networks in this chapter.
5 Simulations on a process with known process parameters

The performance of the three controllers based on neural networks are compared with the performance of the constant extremum controller and the mixed extremum controller. The results of these simulations are shown in this chapter. The controller, that has the best performance, is the one that has the lowest losses. To make it possible to compare the losses for the different controllers, the losses for all the controllers are calculated in the same way. The losses in the simulations are given by formula (3.4) and the calculations of the losses starts at simulation number 1000 to eliminate the effects of any initial conditions, see table 5.6.1. All controllers starts with \( u = u^{\text{start}} = 0 \) and \( y = y^{\text{start}} = 0.25 \) in the simulations and these starting values of \( u \) and \( y \) are also used, when the controllers based on neural networks are trained. All the controllers are simulated during 5000 steps and the controllers based on neural networks are first trained during 30 000 steps, before the simulations can be done. All the controllers are simulated for the standard deviation \( \sigma_y = 0.2 \) and \( \sigma_e = 1 \). A larger and a smaller standard deviation is used, since it makes it easier to analyse the performance of the controllers based on neural networks. All the simulations and the training of the controllers, based on neural networks, are based on the process model and the process parameters given by formula (3.2) and formula (3.3). The same random sequence of errors \( \{ e_i(k) \} \) is used with the variance \( V(e_i(k)) = \sigma^2_i = 1 \), in the process given by formula (3.3), in the simulations. This error vector is implemented with same seed in Matlab and is then multiplied with \( \sigma_e = 0.2 \) or \( \sigma_e = 1 \), to obtain a random sequence \( \{ e(k) \} \) with the standard deviation \( \sigma_e = 0.2 \) or \( \sigma_e = 1 \). In other words an identical error sequence is obtained for \( \sigma_e = 0.2 \) and \( \sigma_e = 1 \) in the simulations of the different controllers.

In table 5.1 important parameters and variables, that are used when the neural networks are trained, can be seen. The first value of the step length is given by \( \eta_{\text{start}} \) and the step length \( \eta \), see formula (4.15), is constantly changing it’s value, when the adaptive step length is used. How the step length changes with the adaptive step length parameters, \( a \) and \( d \), can be seen in formula (2.29). During step 101-1000, when the neural networks are trained, the step length is fixed to \( \eta_{\text{fit1}} \) and during step 1001-50 000 the step length is fixed to \( \eta_{\text{fit2}} \), see table 5.1. The values of the variables \( \eta_{\text{start}} \), \( \eta_{\text{fit1}} \) and \( \eta_{\text{fit2}} \) varies between the different controllers and the different simulations. Note that the same value of the momentum parameter \( \alpha \), see formula (4.14), is used during the entire training and that a value of \( \alpha \) between 0.05 and 0.1 works well. During step 1-100, when the controllers based on neural networks are trained, \( \sigma_e \) is set to 0.85 for the controllers, that are simulated with \( \sigma_e = 1 \), see table 5.1. The reason why \( \sigma_e = 0.85 \) during these steps of training, is to make it easier for the controllers based on neural networks, to find the right ”constant control law” \( u(k) = -0.25 \).
A summary of how the neural networks are trained can be seen. The standard deviation is divided into two columns. One column is used for the simulations with \( \sigma_e = 0.2 \) and the other column is used for the simulations with \( \sigma_e = 1 \). The three controllers based on neural networks are all trained according to this table.

<table>
<thead>
<tr>
<th>The number of steps of training:</th>
<th>The standard deviation, ( \sigma_e ):</th>
<th>The step length, ( \eta ):</th>
<th>The adaptive step length parameters, ( a ) and ( d ):</th>
<th>The momentum parameter, ( \alpha ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1-100 ( \sigma_e = 0.2 ) ( \sigma_e = 0.85 )</td>
<td>( \eta_{start} )</td>
<td>( a = d = 0.05 )</td>
<td>( \alpha = 0.05 )</td>
<td></td>
</tr>
<tr>
<td>Step 101-1000 ( \sigma_e = 0.2 ) ( \sigma_e = 1 )</td>
<td>( \eta_{fix1} )</td>
<td>No adaptive step length is used.</td>
<td>( \alpha = 0.05 )</td>
<td></td>
</tr>
<tr>
<td>Step 1001-3000 ( \sigma_e = 0.2 ) ( \sigma_e = 1 )</td>
<td>( \eta_{fix2} )</td>
<td>No adaptive step length is used.</td>
<td>( \alpha = 0.05 )</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1. Important variables and parameters that are used during the training of the neural networks can be seen.

Once the controllers based on neural networks have been trained, the values of the weight matrices \( w_i^{(k)} \) and the bias \( b_i^{(k)} \) for \( i = 1, 2 \) and \( k = 30 \ 000 \) are saved, see formula (4.1)-(4.12), and used in the simulations. The momentum parameter and the step length are not needed in the simulations. Only the weights and bias are needed to simulate the controllers based on neural networks, once the neural network has been properly trained.
5.1 The constant extremum controller

The constant extremum controller is explained in section 3.4. The constant extremum controller is simulated for $\sigma_e = 0.2$, see graph 5.1.1 and $\sigma_e = 1$, see graph 5.1.2. Note that the control law for this controller is $u(k) = -0.25$.

Graph 5.1.1. The top curve shows the output $y(k + 1)$ and the bottom curve shows the control law $u(k) = -0.25$ for the constant extremum controller. The line in the top curve is set at the set point $y_0 = 4.375$. $\sigma_e = 0.2$ in the simulation.
Graph 5.1.2. The top curve shows the output $y(k + 1)$ and the bottom curve shows the control law $u(k) = -0.25$ for the constant extremum controller. The line in the top curve is set at $y_0 = 4.375$, $\sigma_e = 1$ in the simulation.

The control law for the constant extremum controller is given by a very simple control law, that doesn’t take into account any dynamic effects in the process. The variance of the output, $y(k + 1)$, is therefore given by the open loop variance.
5.2 The mixed extremum controller

The mixed extremum controller is explained in section 3.5. The mixed extremum controller is simulated for $\sigma_e = 0.2$, see graph 5.2.1 and $\sigma_e = 1$, see graph 5.2.2. Note that the control law for this controller is given by (3.25).

Graph 5.2.1. The top curve shows the output $y(k + 1)$ and the bottom curve shows the control law $u(k)$, when the mixed extremum controller is used. The line in the top curve is set at $y_0 = 4.375$ and the line in the lower curve is set at $u_0 = -0.25$. $\sigma_e = 0.2$ in the simulation.

The variance of the output $y(k+1)$, is the open loop variance, when the minimum variance controller in (3.25) can’t be used and the constant extremum controller in (3.25) must be used. A lower variance is obtained when the minimum variance controller in (3.25) can be used. Only when $y(k + 1)$ lies below the set point, the output may be pushed up towards the set point $y_0 = 4.375$ and nothing can be done when $y(k + 1)$ lies above $y_0$, except using the control law $u(k) = -0.25$. The control law for the mixed extremum controller takes into account the dynamic effects in the process. The varying lower curve, in graph 5.2.1, and the curve, in the bottom graph in graph 5.2.2, show the control law for the mixed extremum controller. In the long run the losses for the mixed extremum controller are lower than the losses for the constant extremum controller, see table 5.6.1 and graph 5.3.3.
Graph 5.2.2. The top curve shows the output $y(k + 1)$ and the bottom curve shows the control law $u(k)$, when the mixed extremum controller is used. The line in the top curve is set at $y_0 = 4.375$ and the line in the lower curve is set at $u_0 = -0.25$. $\sigma_e = 1$ in the simulation.
5.3 Controller 1 based on neural networks

A more detailed explanation of the controller 1 based on neural networks is given in section 4.3. The size of the neural network for this controller is given by $N_1 = N_4 = 1$, $N_2 = 25$ and $N_3 = 30$. The input signal $\xi(k)$ for controller 1 is given by $w(k-1)$ and the output is given by $u(k)$. Important parameters for the training of controller 1 can be seen in table 5.1. Note that the structures of controller 1, controller 2 and controller 3 based on neural networks are different. Controller 1 is simulated for $\sigma_e = 0.2$, see graph 5.3.1-5.3.5, and $\sigma_e = 1$, see graph 5.3.6-5.3.8.

Graph 5.3.1. The top curve shows the output $y(k + 1)$ and the bottom curve shows the control law $u(k)$, when controller 1 is used. The line in the top curve is set at $y_0 = 4.375$ and the line in the lower curve is set at $u_0 = -0.25$. $\sigma_e = 0.2$ in the simulation.
Graph 5.3.2. The solid curve is the control signal $u_{\text{mixed}}(k)$ for the mixed extremum controller as a function of $w(k)$. The dashed curve is the control signal $u(k)$ from controller 1 as a function of $w(k)$. The input signal $w(k)$ is chosen to be a linear signal that grows from $-0.6$ to $2$ with the step length $0.05$. $\sigma_e = 0.2$ in the simulation.
Graph 5.3.3. The losses for controller 1 is given by the solid curve and the constant extremum controller is given by the dotted curve. The losses for the mixed extremum controller is given by the dashed dotted curve. $\sigma_e = 0.2$ in the simulation.

In graph 5.3.1-5.3.5 the following values are used on the step length: $\eta_{\text{start}} = 0.01$, $\eta_{\text{fat}} = 0.04$ and $\eta_{\text{fat2}} = 0.05$, see table 5.1. In the lower curve in graph 5.3.1 and in the bottom graph in graph 5.3.6 a varying control signal for controller 1 can be seen. The control signal for this controller is similar to the mixed extremum controller, except when $w(k)$ lies close to 0.125 and when $w(k) < -0.1$, see graph 5.3.2 and graph 5.3.7.
Graph 5.3.4. The weights are plotted during 30 000 steps of training for controller 1. The changes of a diagonal and an off diagonal element of the weight matrices in neuron layer one and two can be seen. The two top graphs shows the changes of two elements of the weight matrix in neuron layer one and the two bottom graphs shows similar elements in neuron layer two, see formula (4.1) and formula (4.7).

From graph 5.3.4 and graph 5.3.5 the changes of a few selected typical weights and bias in neuron layer one and two are plotted during the training of controller 1, when \( \sigma_e = 0.2 \). The weights and the bias are not converging too well, because a two layer neural network can’t follow a discontinuous output signal too well, see formula (3.25). At the ends of the interval of \( w(k) \), the neural networks don’t follow the mixed extremum controller too well, since there is less training data at these values of \( w(k) \). The variance of the output \( y(k+1) \) for controller 1, when \( \sigma_e = 0.2 \) and \( \sigma_e = 1 \), is reduced to a value that lies below the open loop variance, because this controller takes into account the dynamic effects in the process.
Graph 5.3.5. The bias are plotted during 30 000 steps of training for controller 1. The changes of the two first elements of the bias vector in neuron layer one and two can be seen. The two top graphs shows the changes of two elements of the bias vector in neuron layer one and the two bottom graphs shows similar elements in neuron layer two, see formula (4.1) and formula (4.7).

The losses for controller 1, when $\sigma_e = 0.2$, lies somewhere between the losses for the mixed extremum controller and the losses for the constant extremum controller, see graph 5.3.3 and table 5.6.1. The reason why the losses are so large for controller 1, when $\sigma_e = 0.2$, is that the control signal for controller 1 don’t follow the control law for the mixed extremum controller so well in the interval $0.05 \leq w(k) \leq 0.3$. Note that the probability that controller 1, when $\sigma_e = 0.2$, is operating in this interval, is large. Controller 1, when $\sigma_e = 0.2$, is seldom operating in the interval $w(k) \leq -0.25$ and in the interval $w(k) \geq 1$, why it’s not so important that the control law for controller 1 follows the control law for the mixed extremum controller in these intervals. Note that the largest value of $u(k)$ for controller 1 is 0.3 and that the largest value of $u(k)$ for the mixed extremum controller is 0.4, when $\sigma_e = 0.2$. Also when $\sigma_e = 1$, the largest value of $u(k)$ is larger for the mixed extremum controller than for controller 1. This fact, together with graph 5.3.2 and graph 5.3.7, implies that controller 1, when $\sigma_e = 0.2$ and $\sigma_e = 1$, uses a lower value on the control signal than the mixed extremum controller, when the minimum variance controller in formula (3.25) is used.
Graph 5.3.6. The top curve shows the output $y(k+1)$ and the bottom curve shows the control law $u(k)$, when controller 1 is used. The line in the top curve is set at $y_0 = 4.375$ and the line in the lower curve is set at $u_0 = -0.25$. $\sigma_e = 1$ in the simulation.
Graph 5.3.7. The solid curve is the control signal \( u_{\text{mixed}}(k) \) for the mixed extremum controller as a function of \( w(k) \). The lower curve is the control signal \( u(k) \) from controller 1 as a function of \( w(k) \). The input signal \( w(k) \) is chosen to be a linear signal that grows from \(-1.8\) to \(6\) with the step length \(0.05\). \( \sigma_e = 1 \) in the simulation.

In graph 5.3.6-5.3.8 the following values are used on the step length: \( \eta_{\text{short}} = 0.01 \), \( \eta_{\text{fast}} = 0.02 \) and \( \eta_{\text{fast2}} = 0.01 \). For controller 1, it can be seen that the control signal for this controller don’t follow the control signal for the mixed extremum controller too well, in the interval \(0 < w < 0.25\), when \( \sigma_e = 0.2\), and in the interval \(0 < w < 0.8\), when \( \sigma_e = 1\), see graph 5.3.2 and graph 5.3.7. When \( w(k) \leq -0.25\) the mixed extremum controller chooses a control signal \( u(k) \), so that \( p(k+1) = 0\), see formula (3.17). The controller 1 uses a lower value on the control signal \( u(k) \) in this interval, especially when \( \sigma_e = 1\), why a lower value on \( y(k+1) \) is obtained, see graph 5.3.2 and graph 5.3.7.
Graph 5.3.8. The losses for controller 1 are given by the solid curve and the losses for the mixed extremum controller are given by the dashed dotted curve. The losses for the constant extremum controller, which have the largest losses, are given by the dotted curve. $\sigma_e = 1$ in the simulation.

Note that all controllers based on neural networks have chosen a control law, so that lower values on $y(k + 1)$ and $\tilde{y}(k + 1)$, see formula (3.13), are obtained for the controllers based on neural networks than for the mixed extremum controller, when the minimum variance controller is used. In other words the probability that $\tilde{y}(k + 1)$ is lower than the set point $y_o$, is larger for the controllers based on neural networks than for the mixed extremum controller, when the minimum variance controller can be used. This fact makes it possible to use the minimum variance controller more frequently for the controllers based on neural networks than for the mixed extremum controller, see formula (3.25). From this fact it can be seen that it isn’t so important for controller 1, to exactly follow the mixed extremum controller, when $w(k) \leq -0.25$. In the long run the losses for controller 1 are lower than the losses for the mixed extremum controller in the interval $w(k) \leq -0.25$. The control signal $u(k)$ for controller 1, when $\sigma_e = 1$, operates in the interval $-2 \leq w(k) \leq 4$, see graph 5.3.7. Values of $w(k)$ outside this interval are seldom used in controller 1, why the corresponding control law for controller 1, in this interval of $w(k)$, is not so important. The lowest losses are obtained for controller 1, when $\sigma_e = 1$, see graph 5.3.8 and table 5.6.1. Note that the largest losses are obtained for the constant extremum controller, when $\sigma_e = 0.2$ and $\sigma_e = 1$, in all simulations, see table 5.6.1.
5.4 Controller 2 based on neural networks

A more detailed explanation of the controller 2 based on neural networks is given in section 4.4. The size of the neural network for this controller is given by $N_1 = 5$, $N_2 = 25$, $N_3 = 30$ and $N_4 = 1$. The input vector $\xi^{(k)}$ for controller 2 is given by $y(k-1)$, $u(k-2)$, $u(k-1)$, $u^2(k-1)$ and $y(k)$ and the output is given by $u(k)$. Important parameters for the training of controller 2 can be seen in table 5.1. Controller 2 is simulated for $\sigma_e = 0.2$, see graph 5.4.1 and graph 5.4.2, and $\sigma_e = 1$, see graph 5.4.3 and graph 5.4.4.

Graph 5.4.1. The top curve shows the output $y(k+1)$ and the bottom curve shows the control law $u(k)$, when controller 2 is used. The top line is set at $y_0 = 4.375$ and the bottom line is set at $u_0 = -0.25$. $\sigma_e = 0.2$ in the simulation.
Graph 5.4.2. The losses for controller 2 are given by the solid curve and the losses for the mixed extremum controller are given by the dashed dotted curve. The losses for the constant extremum controller, which have the largest losses, are given by the dotted curve. $\sigma_e = 0.2$ in the simulation.

In graph 5.4.1 and 5.4.2 the following values are used on the step length: $\eta_{stt} = 0.01$, $\eta_{sr1} = 0.09$ and $\eta_{sr2} = 0.1$. The control law for controller 2 is given by the lower varying curve in graph 5.4.1 and the curve in the bottom graph in graph 5.4.3. When $\sigma_e = 0.2$, the largest losses are obtained for the constant extremum controller and the lowest losses are obtained for the mixed extremum controller. The losses for controller 2, when $\sigma_e = 0.2$, are somewhat larger than the losses for the mixed extremum controller, see graph 5.4.2 and table 5.6.1. Among the controllers based on neural networks, controller 2 has the lowest losses, when $\sigma_e = 0.2$. 
Graph 5.4.3. The curve in the top graph shows the output $y(k + 1)$ and the curve in the bottom graph shows the control law $u(k)$, when controller 2 is used. The line in the top graph is set at $y_0 = 4.375$ and the line in the bottom graph is set at $u_0 = -0.25$. $\sigma_e = 1$ in the simulation.

In graph 5.4.3 and 5.4.4 the following values are used on the step length: $\eta_{\text{init}} = 0.03$, $\eta_{\text{for1}} = 0.03$ and $\eta_{\text{for2}} = 0.03$. Note that controller 2, when $\sigma_e = 1$, is the only controller that uses a control law that is similar to the control law for the mixed extremum controller with the minus sign in formula (3.25), when the minimum variance controller can be used. The losses for controller 2, when $\sigma_e = 1$, are larger than the losses for the mixed extremum controller and are lower then the losses for the constant extremum controller, see graph 5.4.4 and table 5.6.1. Among the controllers based on neural networks, controller 2 has the largest losses, when $\sigma_e = 1$. 
Graph 5.4.4. The losses for controller 2 are given by the solid curve and the losses for the mixed extremum controller are given by the dashed dotted curve. The losses for the constant extremum controller, which have the largest losses, are given by the dotted curve. $\sigma_v = 1$ in the simulation.

Note that a complex pattern exist between the variables in the input vector $\xi^{(k)}$ and the control signal $u(k)$ for controller 2 and controller 3. This fact makes it possible for controller 2 and controller 3 to find a control law $u(k)$, that takes into account values on the earlier control signals $u(k-1)$ and $u(k-2)$, the output signal $y(k)$ and the predicted value of $y(k-1)$, $\hat{y}(k-1)$, see formula (3.13). When the minimum variance controller can be used in controller 2 and controller 3, a control law may be chosen, that decreases the losses more than the mixed extremum controller in the long run.
5.5 Controller 3 based on neural networks

A more detailed explanation of the controller 3 based on neural networks is given in section 4.5. Controller 3 has the most complex structure of all the controllers that are based on neural networks in this chapter. The size of the neural network for this controller is given by $N_1 = 5$, $N_2 = 25$, $N_3 = 30$ and $N_4 = 1$. The input vector $\xi^{(k)}$ for controller 3 is given by $\hat{y}(k-1)$, $u(k-2)$, $u(k-1)$, $u^2(k-1)$ and $y(k)$ and the output is given by $u(k)$. Important parameters for the training of controller 3 can be seen in table 5.1. This controller has a lot in common with controller 2. Controller 3 is simulated for $\sigma_e = 0.2$, see graph 5.5.1 and graph 5.5.2, and $\sigma_e = 1$, see graph 5.5.3 and graph 5.5.4.

![Graph 5.5.1](image)

**Graph 5.5.1.** The top curve shows the output $y(k+1)$ and the bottom curve shows the control law $u(k)$, when controller 3 is used. The top line is set at $y_0 = 4.375$ and the bottom line is set at $u_0 = -0.25$. $\sigma_e = 0.2$ in the simulation.
Graph 5.5.2. The losses for controller 3 are given by the solid curve and the losses for the mixed extremum controller are given by the dashed dotted curve. The losses for the constant extremum controller, which have the largest losses, are given by the dotted curve. $\sigma_e = 0.2$ in the simulation.

In graph 5.5.1 and 5.5.2 the following values are used on the step length: $\eta_{0.005}$, $\eta_{0.04}$ and $\eta_{0.04} = 0.04$. The control law for controller 3 is given by the lower varying curve in graph 5.5.1 and the curve in the bottom graph in graph 5.5.3. When $\sigma_e = 0.2$, the largest losses are obtained for the constant extremum controller and the lowest losses are obtained for the mixed extremum controller. The losses for controller 3, when $\sigma_e = 0.2$, are somewhat larger than the losses for the mixed extremum controller, see graph 5.5.2 and table 5.6.1.
Graph 5.5.3. The curve in the top graph shows the output $y(k+1)$ and the curve in the bottom graph shows the control law $u(k)$, when controller 3 is used. The line in the top graph is set at $y_0 = 4.375$ and the line in the bottom graph is set at $u_0 = -0.25$. $\sigma_c = 1$ in the simulation.
Graph 5.5.4. The losses for controller 3 are given by the solid curve and the losses for the mixed extremum controller are given by the dashed dotted curve. The losses for the constant extremum controller, which have the largest losses, are given by the dotted curve. $\sigma_e = 1$ in the simulation.

In graph 5.5.3 and 5.5.4 the following values are used on the step length: $\eta_{\text{start}} = 0.005$, $\eta_{\text{fa1}} = 0.02$ and $\eta_{\text{fa2}} = 0.01$. The losses for controller 3, when $\sigma_e = 1$, are somewhat larger than the losses for the mixed extremum controller and lower than the losses for the constant extremum controller, see graph 5.5.4 and table 5.6.1.
5.6 Conclusions

The important results from the graphs are summarised in this section. The losses for the different controllers, when $\sigma_e = 0.2$ and $\sigma_e = 1$ during the simulations, can be seen in table 5.6.1 and the mean value of $w(k)$, $E(w)$, for the mixed extremum controller and the three controllers based on neural networks can be seen in table 5.6.2. By comparing table 5.6.1 and table 5.6.2, it can be seen that a connection exist between the losses and $E(w)$. The lower the value of $E(w)$, the lower the losses are for the controllers based on neural networks. Note that the three controllers based on neural networks operates mainly in the interval $-0.3 \leq w(k) \leq 1$, when $\sigma_e = 0.2$, and mainly in the interval $-1.5 \leq w(k) \leq 4$, when $\sigma_e = 1$. The results of the simulations are that the three controllers based on neural networks have a good performance, when $\sigma_e = 0.2$ and $\sigma_e = 1$. With the exception of controller 1, when $\sigma_e = 1$, the three controllers based on neural networks, when $\sigma_e = 0.2$ and $\sigma_e = 1$, have larger losses than the mixed extremum controller and lower losses than the constant extremum controller, see table 5.6.1. The mixed extremum controller has the lowest losses, when $\sigma_e = 0.2$, and the lowest losses are obtained for controller 1, when $\sigma_e = 1$. Note that the losses for controller 1 are just a little lower than the losses for the mixed extremum controller, when $\sigma_e = 1$. The largest losses are obtained for the constant extremum controller, when $\sigma_e = 0.2$ and $\sigma_e = 1$. Among the controllers based on neural networks, when $\sigma_e = 0.2$, controller 2 has the best performance and controller 1 has the worst performance. By making a similar comparison between the controllers based on neural networks, when $\sigma_e = 1$, it can be seen that controller 2 has the largest losses, see table 5.6.1. If only one of the controllers based on neural networks are to be used, both for $\sigma_e = 0.2$ and $\sigma_e = 1$, it's recommended to use controller 3. The conclusion that can be drawn from the simulations in this chapter is that the three controllers based on neural networks work exceptionally well, with the exception of controller 1, when $\sigma_e = 0.2$, and controller 2, when $\sigma_e = 1$.

<table>
<thead>
<tr>
<th>The standard deviation $\sigma_e$</th>
<th>The losses for the constant extremum controller:</th>
<th>The losses for the mixed extremum controller:</th>
<th>The losses for controller 1 based on neural networks:</th>
<th>The losses for controller 2 based on neural networks:</th>
<th>The losses for controller 3 based on neural networks:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_e = 0.2$</td>
<td>524.56</td>
<td>460.93</td>
<td>480.27 (*477.71)</td>
<td>464.70</td>
<td>466.05</td>
</tr>
<tr>
<td>$\sigma_e = 1$</td>
<td>$1.3114 \cdot 10^4$</td>
<td>$1.1841 \cdot 10^4$</td>
<td>$1.1825 \cdot 10^4$</td>
<td>$1.2205 \cdot 10^4$</td>
<td>$1.1870 \cdot 10^4$</td>
</tr>
</tbody>
</table>

* This result is obtained, when controller 1 is trained 100 000 steps.

Table 5.6.1. The losses for the constant extremum controller, the mixed extremum controller and the three controllers based on neural networks are shown.
The mean value of \( w(k) \), \( E(w) \), is calculated for the mixed extremum controller and the three controllers based on neural networks. Values of \( w(k) \) from simulation number 1000 to simulation number 5000 are used to obtain \( E(w) \). The calculations start at simulation number 1000 to eliminate the effects of any initial conditions.

<table>
<thead>
<tr>
<th>The standard deviation, ( \sigma_e ):</th>
<th>( E(w) ) for the mixed extremum controller:</th>
<th>( E(w) ) for controller 1 based on neural networks:</th>
<th>( E(w) ) for controller 2 based on neural networks:</th>
<th>( E(w) ) for controller 3 based on neural networks:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma_e = 0.2 )</td>
<td>0.26687</td>
<td>0.26762 (* 0.26703)</td>
<td>0.22936</td>
<td>0.24637</td>
</tr>
<tr>
<td>( \sigma_e = 1 )</td>
<td>1.03564</td>
<td>0.91310</td>
<td>1.04814</td>
<td>1.01590</td>
</tr>
</tbody>
</table>

* This result is obtained, when controller 1 is trained 100000 steps.

Table 5.6.2. By comparing this table with table 5.6.1 a connection can be seen between the losses and \( E(w) \). For the controllers based on neural networks the losses decrease, when \( E(w) \) decreases.

To understand why the controllers based on neural networks don’t work so well, when \( w(k) \) lies close to 0.125, it’s necessary to study how the weight matrices and the bias are updated in the neural networks, see section 4.3-4.5. The changes in the weight matrices and in the bias are proportional to \( \frac{\partial E(k + 1)}{\partial \chi^{(k)}} \), see formula (4.41) for controller 1 and controller 2, and formula (4.43) for controller 3. The term \( \frac{\partial E(k + 1)}{\partial \chi^{(i)}} \) consists of mainly two term that are multiplied with each other. The term \( \frac{\partial E(k + 1)}{\partial \chi^{(i)}} \) goes to zero, when either \( p(k + 1) \) goes to zero or when the term in the brackets goes to zero in formula (4.41) and formula (4.43). These solutions corresponds to the "minimum variance controller" and the "constant extremum controller", with the control law \( u(k) = -0.25 \), see formula (3.25). Close to \( w(k) = 0.125 \) the controllers based on neural networks can’t decide which solution gives the lowest value on the network error \( E(k + 1) \), see formula (4.38). This fact implies that it’s difficult for the controllers based on neural networks to follow the control law that is given by the mixed extremum controller, when \( w(k) \) lies close to 0.125. Note that when \( \frac{\partial E(k + 1)}{\partial \chi^{(i)}} \neq 0 \) the neural network is no longer updated and therefore the neural network is no longer trained. The weight matrices and the bias are then fixed, since the updates of the neural network don’t change the weight matrices and the bias in the neural network, see section 4.3-4.5.
6 Summary and conclusions

Two controllers in computer controlled systems, the constant extremum controller and the mixed extremum controller, are compared with three controllers, that are based on neural networks, when $\sigma_e = 0.2$ and $\sigma_e = 1$, see table 5.6.1. All controllers, based on neural networks, have losses that are larger than the mixed extremum controller and lower losses than the constant extremum controller, when $\sigma_e = 0.2$ and $\sigma_e = 1$, with the only exception of controller 1, when $\sigma_e = 1$. The mixed extremum controller have somewhat larger losses than controller 1, when $\sigma_e = 1$. The three controllers, that are based on neural networks, work exceptionally well, with the exception of controller 2, when $\sigma_e = 1$, and controller 1, when $\sigma_e = 0.2$. Among the three controllers, that are based on neural networks, controller 2 has the lowest losses, when $\sigma_e = 0.2$, and controller 1 has the lowest losses, when $\sigma_e = 1$. If only one of the controllers, that are based on neural networks, can be used, when $\sigma_e = 0.2$ and $\sigma_e = 1$, it's recommended to use controller 3.

For the three controllers, that are based on neural networks, a relation can be seen between the losses and the mean value of $w$, $E(w)$, see table 5.6.1 and table 5.6.2. The lower $E(w)$ is, the lower the losses are for the three controllers, that are based on neural networks.

All the controllers, that are based on neural networks, have difficulties in following the control signal $u(k)$, that is given by the mixed extremum controller. It comes from the fact that a discontinuity appears in $u(k)$, when $w(k) = 0.125$, see formula (3.25). A neural network, that has two neuron layers, that are updated by the backpropagation algorithm, can’t follow a discontinuous output signal too well, see section 2.1. Although my neural networks in chapter 4 have three neuron layers, only the two first neuron layers are updated by the backpropagation algorithm, when it’s trained. This means that my neural networks in chapter 4 operates as a neural network, with only two neuron layers. The problems with the controllers, that use the backpropagation algorithm, are that they converge towards a solution very slowly and that they are very sensitive to the step length, see section 2.1, section 2.4 and Bistedt (94) [5] p. 41-42.

Let the term an active neuron layer mean that it is updated by the backpropagation algorithm, during the training. A neural network with four neuron layers, that uses three active neuron layers and the output layer, might work very well as controllers in extremal seeking control, but the time to train the neural networks are even longer than for my controllers in chapter 4. The advantage of using a neural network with four neuron layers, is that the number of neurons in the three active neuron layers can be chosen arbitrarily. Another problem with a neural network, that updates three neuron layers, when it’s trained, is that it’s more sensitive to the step length than my controllers in chapter 4, see Demuth and Beale (93) [4] p. 5.10-5.11 and Bistedt (94) [5] p. 41-42.

In this project it’s been assumed that the process parameters in formula (3.3), $\{a_1, b_9, b_{10}, b_{11}, b_{20}$ and $c_i\}$, are known. If the process parameters are unknown, they can be approximated by an extended least square method, see Wittenmark and Urquhart (95) [4] p. 1642-1644. The estimated process parameters are then used instead of the true parameters.
Bibliography


