Lecture Notes

Linear and Nonlinear System Identification

Herbert Werner

TUHH
Hamburg University of Technology

ICS
Institute of Control Systems

ver. November 17th, 2020

Contents

Introduction **1**

1 Least-Squares Estimation **4**

2 Prediction Error Method **15**

3 Multilayer Perception Networks **27**

4 Solving the Estimation Problem **40**

5 Training Neural Networks Levenberg-Marquardt Backpropagation **51**

6 Predictive Control Using Neural Networks **62**

7 Linear Subspace Identification **72**

Bibliography **82**

A APPENDICES **83**

A Solutions to Exercises **84**

Introduction

This lecture course *Linear and Nonlinear System Identification* provides an introduction to a field that is of considerable importance for the solution of practical control problems when a model-based controller design is involved (as it usually is). It is concerned with obtaining a model of the plant to be controlled when a first-principles model is hard to obtain or would not lend itself easily to controller design because of its structure or complexity. An alternative to first-principles modeling (that can serve as a complement when an existing physical model is too complex for control synthesis) is experimental system identification, where numerical optimisation is used to fit the parameters of a predetermined model structure to experimentally obtained input and output data. The chosen model structure may be motivated by physical insight into the plant dynamics (in that case we speak of a *grey-box model*, in contrast to first-principles models which are referred to as *white-box models*). Often the choice is just dictated by the need for a model that is simple enough to be employed in standard model-based controller synthesis procedures. When linear control techniques are used, that typically means linear state space or transfer function models. Since such models are not based on physical insight, they are referred to as *black-box models*. In this course we will focus on the identification of black-box models.

In black-box identification the choice of the model structure represents a trade-off between on one hand the complexity of the model and the ensuing control synthesis procedure, and on the other hand the desired accuracy and control performance. A fundamental decision to be made is that between a linear and a nonlinear model structure. In practice linear models are preferred, because powerful control techniques are available for such models. But there are critical applications where nonlinear plant dynamics are too dominant for a linear approximation to be acceptable. In this course we will cover the identification of both linear and nonlinear model structures. We will introduce a framework that unifies the identification of a variety of different model structures, linear and nonlinear. This framework is known as the *Prediction Error Method* and forms the basis of most identification techniques that are employed in practical applications: the chosen model structure is transformed into a *predictor model*, a model that takes the experimental input-output data up to a given time as its input and generates a one-step-ahead prediction of the output. The parameters of the predictor model (which are the desired model parameters) can then be found by solving the problem of minimising the sum of squared prediction

errors.

Before presenting the framework of the prediction error method, we show in Chapter 1 that in the simplest case - a linear model structure and output measurements corrupted only by white noise filtered through the plant dynamics - the identification problem can be solved as a linear least squares problem. The price for being able to solve this problem simply by solving a linear equation - the *normal equation* - is that the model structure does not allow any information about noise and perturbations to be extracted and exploited for improving the model. This model structure is known as ARX model and appears again in Chapter 2 as a special case; it is the simplest of a number of linear model structures which differ in the assumptions on the noise process that corrupts the measurements. In order to illustrate the trade-off between model complexity and model accuracy when choosing a model structure, we discuss two model structures in detail: ARX models and ARMAX models; the latter structure allows to include information about the noise in the model. In all model structures except ARX the one-step-ahead prediction is nonlinear in the model parameters, and the estimation problem needs to be solved iteratively. The gradient-based iterative solution of the estimation problem is discussed in Chapter 4.

Identification of nonlinear dynamic models is considered in Chapter 5; here rather than providing an overview of existing nonlinear system identification techniques, we will focus on one particular approach: training a neural network - more specifically a *multilayer perceptron network* introduced in Chapter 3 - to capture the dynamic behaviour of a nonlinear system. It is shown how this can be done within the framework of the prediction error method, and we will see that the same trade-off between model complexity and accuracy arises that is encountered for linear model structures. Following the discussion of linear system identification, the neural network based NNARX and NNARMAX structures are considered.

To motivate the representation of nonlinear systems by neural network models, we give in Chapter 6 a brief introduction to predictive control. While the presentation of an approach known as *Generalised Predictive Control*, which is based on a linear model structure, should be of interest in its own right, we show how a neural network model of a nonlinear system together with the so-called *instantaneous linearisation* technique can form the basis of a very efficient predictive control scheme for nonlinear systems. The discussion of nonlinear control is heuristic in the sense that stability of the closed-loop system is not guaranteed under the proposed scheme. This is however in line with industrial practice, where in process control predictive schemes have been used for decades efficiently in spite of a lack of stability guarantees. A more rigorous treatment of nonlinear control that includes an analysis of closed-loop stability and performance is provided in the lecture course *Advanced Topics in Control*.

The model structures considered in the course material outlined so far are restricted to single-input single-output systems; for linear model structures the identified model parameters can be seen as polynomial coefficients in transfer function models. Even

though it is possible to extend these model structures to multi-input multi-output systems, it is usually more convenient to handle MIMO systems in state space form. To complete this course, Chapter 7 gives a brief introduction to a technique that is not based on the prediction error method and can be used to identify state space models of linear SISO and MIMO systems; this approach is known as *subspace identification*.

Whereas the lecture material emphasizes theoretical concepts involved in system identification, some practical issues such as experiment design and preprocessing of data are covered in a series of exercise problems, which are to be solved using Matlab tools including the System Identification Toolbox.

Chapter 1

Least-Squares Estimation

All methods for designing and analysing control systems that have been introduced in our earlier control courses are *model based*, i.e. it is assumed that a dynamic plant model in the form of a transfer function or state space realization is available. In practice, obtaining such a model can take up a significant part of the time required for solving a given control problem. Physical modelling can become difficult if the plant dynamics are complex and not well understood. Even when it is possible to obtain a model from first principles, it may not be suitable for controller design because of its structure or complexity. An alternative is to obtain a plant model experimentally by measuring the response of the plant to suitable test signals; this approach is known as *system identification*. Because no physical insight into the plant behaviour is utilized, this method is also referred to as *black box modelling*. One of the advantages of black-box modeling is that one can impose a structure and complexity on the model that makes it a suitable basis for controller design. This chapter shows how parameters of a model with predefined structure and order can be determined by solving a least squares problem.

Transfer functions and state space models are called *parametric models*, because the complete information about the dynamic behaviour of the system is contained in a fixed number of parameters - e.g. the coefficients of the transfer function polynomials or the entries of the matrices of a state space model. *Nonparametric models* on the other hand are representations of plant dynamics that cannot be expressed by a finite number of parameters, such as the shape of the frequency response or the impulse response of the system. In this course we will consider the experimental identification of parametric plant models. Because the input-output data used for system identification are usually sampled-data sequences, the identified models are discrete-time models. After introducing the concept of a linear regression model in Section 1, we will discuss the identification of transfer function models for SISO systems in Section 1.

Least Squares Estimation

Assume that we are observing a process - characterized by a quantity $y(t)$ - at time instants $t = 0, T, 2T, \ldots$ where T is a given sampling time. Assuming for simplicity $T = 1$, this observation yields a data sequence $y(k),\ k = 0, 1, 2, \ldots$ We further assume that the process variable $y(k)$ at time instant k depends linearly on the values of certain other variables $\varphi_1(k),\ \varphi_2(k), \ldots,\ \varphi_n(k)$ which are known and available at the same time. A linear process model is then

$$y(k) = \varphi_1(k)\theta_1 + \varphi_2(k)\theta_2 + \ldots + \varphi_n(k)\theta_n + \varepsilon(k) \tag{1.1}$$

where the dependence of the quantity y on the measured variables is determined by the parameters $\theta_1,\ \theta_2, \ldots,\ \theta_n$. The term

$$\varepsilon(k) = y(k) - \varphi_1(k)\theta_1 - \varphi_2(k)\theta_2 - \ldots - \varphi_n(k)\theta_n$$

is added to allow for modelling errors, e.g. measurement errors or inaccurate knowledge of values of the parameters θ_i. Modelling errors can also arise if the true process depends on the measured variables in a nonlinear way, or if it depends on additional variables that are not included in the above model.

The model (1.1) can be written in a more compact form as a linear regression model

$$y(k) = [\varphi_1(k) \ldots \varphi_n(k)] \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} + \varepsilon(k) = \varphi^T(k)\theta + \varepsilon(k) \tag{1.2}$$

where two column vectors - the vector of regression variables $\varphi(k)$ and the parameter vector θ - have been introduced. Given a set of measured data $y(l)$ and $\varphi(l)$, $l = 1, 2, \ldots, N$, we can now pose the *least squares estimation problem*: find the parameter vector θ that best fits the data, in the sense that the sum of the squared errors

$$V(\theta) = \sum_{l=0}^{N} e^2(l) \tag{1.3}$$

is minimized.

Example 1.1

Suppose a transfer function model of an unknown system is to be identified which is thought to be governed by a first order linear difference equation. Assume that a measured data set $\{y(l), u(l)\},\ l = 1, 2, \ldots, N$ is available. The task is then to find parameters a and b of the transfer function

$$G(z) = \frac{bz^{-1}}{1 + az^{-1}}$$

that lead to a good fit between measured data and output data predicted by the transfer function model.

The unknown system can be expressed in time domain in terms of $G(z)$ in the form of a linear regression model by writing the difference equation of the system as

$$\begin{aligned} y(k) &= -ay(k-1) + bu(k-1) + \varepsilon(k) \\ &= [-y(k-1) \;\; u(k-1)] \begin{bmatrix} a \\ b \end{bmatrix} + \varepsilon(k) \\ &= \varphi^T(k)\theta + \varepsilon(k) = \hat{y}(k) + \varepsilon(k) \end{aligned}$$

where $\varepsilon(k)$ is the modelling error and $\hat{y}(k)$ represents the output predicted by the model.

Solving the Least Squares Estimation Problem

If a data set $\{y(l), u(l)\},\ l = 0, 1, 2, \ldots, N$ is available, we can arrange it in the form

$$Y = \Phi\theta + E \tag{1.4}$$

where we define

$$Y = \begin{bmatrix} y(0) \\ \vdots \\ y(N) \end{bmatrix}, \quad \Phi = \begin{bmatrix} \varphi^T(0) \\ \vdots \\ \varphi^T(N) \end{bmatrix}, \quad E = \begin{bmatrix} \varepsilon(0) \\ \vdots \\ \varepsilon(N) \end{bmatrix}$$

Similarly, we introduce the vector

$$\hat{Y} = \Phi\theta$$

of predicted outputs. Assuming initially that the unknown system can indeed be accurately represented by $G(z)$, and that there are no measurement errors, the modelling error will be zero if the parameter vector θ takes its "true" value: in that case we have $\hat{Y} = Y$, or

$$\Phi\theta = Y \tag{1.5}$$

If we have more measurements available than model parameters, i.e. if $N > n$, this is an overdetermined system of equations. Multiplying from left by Φ^T yields

$$\Phi^T\Phi\theta = \Phi^T Y \tag{1.6}$$

This equation is called the *normal equation* associated with the given estimation problem. If Φ has full column rank, the matrix $\Phi^T\Phi \in \mathbb{R}^{n\times n}$ is non-singular, and we can compute

$$\theta = (\Phi^T\Phi)^{-1}\Phi^T Y \tag{1.7}$$

However, a solution to (1.6) will exist and the parameter vector θ obtained from (1.7) will satisfy (1.5) only if the system is indeed exactly governed by a linear difference equation of the assumed order, and if there are no measurement errors. In real life problems, neither condition will be met, so that θ will not satisfy (1.5) but only (1.4) with $E \neq 0$. The best

we can then achieve is to find the parameter vector θ that is associated with the "smallest modelling error" - in other words the closest approximation we can get with this model in the presence of measurement errors. The following result is derived in Exercise 1.1.

Theorem 1.1

The sum of square errors $V(\theta)$ (1.3) is minimized if the parameter vector satisfies the normal equation (1.6).

If the matrix $\Phi^T\Phi$ is nonsingular, the minimizing parameter vector is given by (1.7).

Geometric Interpretation

A geometric interpretation of the normal equation goes as follows. Rewrite (1.4) as

$$E = Y - \Phi\theta$$

or

$$\begin{bmatrix} \varepsilon(0) \\ \vdots \\ \varepsilon(N) \end{bmatrix} = \begin{bmatrix} y(0) \\ \vdots \\ y(N) \end{bmatrix} - \begin{bmatrix} \varphi_1(0) \\ \vdots \\ \varphi_1(N) \end{bmatrix} \theta_1 - \ldots - \begin{bmatrix} \varphi_n(0) \\ \vdots \\ \varphi_n(N) \end{bmatrix} \theta_n$$

Introducing the column vectors

$$\varphi_i = \begin{bmatrix} \varphi_i(0) \\ \vdots \\ \varphi_i(N) \end{bmatrix}, \qquad i = 1, \ldots, n$$

we have

$$E = Y - \varphi_1\theta_1 - \ldots - \varphi_n\theta_n$$

If the true system can be accurately described by the assumed linear model and if there are no measurement errors, then Y would be in the space spanned by the vectors $\varphi_1 \ldots \varphi_n$. In real life, unmodelled features of the system and measurement errors will in general lead to a vector Y that is outside the data space. The estimation problem can then be interpreted as searching for the linear combination of the vectors $\varphi_1 \ldots \varphi_n$ that comes closest to the vector Y, i.e. that minimizes the squared error E^TE. This is illustrated in Fig. 1.1 for the special case $n = 2$: what we are looking for is the *projection* $\hat{Y}$ of the vector Y onto the space (a plane if $n = 2$) spanned by the measurement vectors φ_i, and $\hat{Y}$ is the vector closest to Y if the error E is orthogonal to this space (plane). But E is orthogonal to this space if it is orthogonal to each of the measurement vectors, i.e. if it satisfies

$$\varphi_i^T E = 0, \qquad i = 1, \ldots, n$$

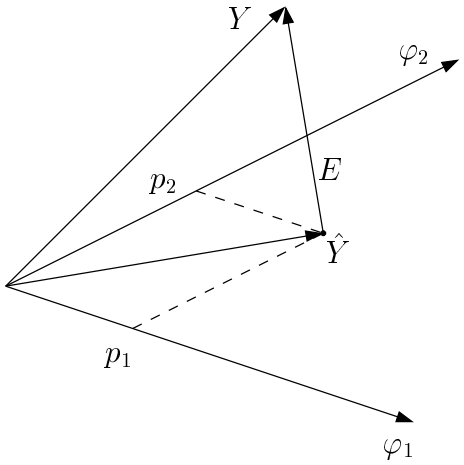

Figure 1.1: Geometric interpretation of the least squares estimation problem

This can be written in a more compact form as $\Phi^T E = 0$ or

$$\Phi^T(Y - \Phi\theta) = 0$$

which is just the normal equation (1.6).

Estimation of Transfer Function Models

We will now apply the idea of least squares estimation to identify a transfer function model of a system from measured input and output data. Thus, assume that data sequences $\{u(0), \ldots, u(N)\}$ and $\{y(0), \ldots, y(N)\}$ are available and that we want to find the pulse transfer function (with specified order n of numerator and denominator polynomial) that gives the best fit between input and output data. We will initially assume for simplicity that the system to be identified can be modelled by the difference equation

$$\hat{y}(k) = b_1 u(k-1) + b_2 u(k-2) + \ldots + b_n u(k-n) \tag{1.8}$$

which means there is no autoregressive component in the output (the a_i's are assumed to be zero) - we will later remove this assumption. The difference equation can be written in regressor form as

$$\hat{y}(k) = [u(k-1)\ u(k-2)\ \ldots\ u(k-n)] \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \varphi^T(k)\theta$$

The measurement matrix Φ takes the form

$$\Phi = \begin{bmatrix} u(n-1) & u(n-2) & \ldots & u(0) \\ u(n) & u(n-1) & \ldots & u(1) \\ \vdots & & & \vdots \\ u(N-1) & u(N-2) & \ldots & u(N-n) \end{bmatrix}$$

where we take $\varphi^T(n)$ as the first row, and we have

$$\Phi^T\Phi = \begin{bmatrix} \sum_{i=n-1}^{N-1} u_i^2 & \sum_{i=n-1}^{N-1} u_i u_{i-1} & \dots & \sum_{i=n-1}^{N-1} u_i u_{i-n+1} \\ \sum_{i=n-1}^{N-1} u_{i-1} u_i & \sum_{i=n-1}^{N-1} u_{i-1}^2 & \dots & \sum_{i=n-1}^{N-1} u_{i-1} u_{i-n+1} \\ \vdots & & \ddots & \vdots \\ \sum_{i=n-1}^{N-1} u_{i-n+1} u_i & \sum_{i=n-1}^{N-1} u_{i-n+1} u_{i-1} & \dots & \sum_{i=n-1}^{N-1} u_{i-n+1}^2 \end{bmatrix}$$

where we used the shorthand notation u_i for $u(i)$. For a solution (1.7) to the estimation problem to be defined, this $n \times n$ matrix needs to be invertible. This requirement places a condition on the input sequence $\{u(1), \dots, u(k)\}$. For example, it is obvious that with a constant input sequence $\{1, \dots, 1\}$ the rank of $\Phi^T\Phi$ will be one and a solution for a model with more than one estimated parameter will in general not exist. To explore this further, we will use the *empirical autocorrelation* of the data sequence $\{u(k)\}$, defined as

$$c_{uu}(l) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N-1} u(i)u(i-l)$$

Introducing the matrix

$$C_{uu}(n) = \begin{bmatrix} c_{uu}(0) & c_{uu}(1) & \dots & c_{uu}(n-1) \\ c_{uu}(1) & c_{uu}(0) & \dots & c_{uu}(n-2) \\ \vdots & & \ddots & \vdots \\ c_{uu}(n-1) & c_{uu}(n-2) & \dots & c_{uu}(0) \end{bmatrix}$$

we find that

$$\lim_{N\to\infty} \frac{1}{N} \Phi^T\Phi = C_{uu}(n)$$

Thus, for sufficiently long data sequences (when the end effects can be neglected and we can consider all sums as taken from 1 to N) we may interpret the matrix $\Phi^T\Phi$ as a scaled version of the empirical covariance matrix $C_{uu}(n)$ of the input signal.

Persistent Excitation

The condition that the matrix $\Phi^T\Phi$ must have full rank is called an *excitation condition* - the input signal must be *sufficiently rich* to excite all dynamic modes of the system. We have seen that for long data sequences we can consider the matrix $C_{uu}(n)$ instead of $\Phi^T\Phi$. The following definition provides a measure for the richness of an input signal.

Definition 1.1

A signal $u(k)$ is said to be persistently exciting of order n if the matrix $C_{uu}(n)$ is positive definite.

The next result is useful for checking whether a signal is persistently exciting of a given order.

Theorem 1.2

A signal $u(k)$ is persistently exciting of order n if and only if

$$\lim_{N\to\infty} \frac{1}{N} \sum_{l=0}^{N-1} \Big(a(z)u(l)\Big)^2 > 0 \quad \forall a(z):\ \deg a(z) \leq n-1 \tag{1.9}$$

Here $a(z)$ is a polynomial in the *forward shift operator* z, i.e.

$$a(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_{n-1} z^{n-1}$$

Recall that the forward shift operator is defined by

$$zu(l) = u(l+1)$$

With these definitions, multiplying a signal $u(l)$ by $a(z)$ yields

$$a(z)u(l) = a_0 u(l) + a_1 u(l+1) + a_2 u(l+2) + \ldots + a_{n-1} u(l+n-1)$$

It is straightforward to prove the above Theorem by observing that the sum on the left hand side of the inequality can be rewritten as

$$\lim_{N\to\infty} \frac{1}{N} \sum_{l=0}^{N-1} \big(a(z)u(l)\big)^2 = a^T C_{uu}(n) a$$

where $a^T = [a_{n-1} \;\; a_{n-2} \;\; \ldots \;\; a_0]$

Theorem 1.2 can be used to determine an upper bound on the order of the persistent excitation of a given signal: if one can find a polynomial $a(z)$ of order n that does not satisfy (1.9), then the signal is *not* persistently exciting (PE) of order n. This idea can be used to show that

- an impulse $\delta(k)$ is PE of order 0
- a step function $\sigma(k)$ is PE of order 1
- a sine wave is PE of order 2
- white noise is PE of any order

For step functions, sine waves and white noise, this is discussed in Exercises 1.2, 1.3 and 1.4. White noise is commonly used as test input when a linear model is to be identified experimentally.

ARX Models

The model (1.8) was based on the assumption that the present output does not depend on past outputs, i.e. there is no autoregressive component in the output. We now remove this assumption and consider the model

$$\hat{y}(k) = -a_1 y(k-1) - \ldots - a_n y(k-n) + b_1 u(k-1) + \ldots b_n u(k-n) \tag{1.10}$$

$$= [-y(k-1) \ \ldots \ -y(k-n) \ \ u(k-1) \ \ldots \ u(k-n)] \begin{bmatrix} a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \tag{1.11}$$

which corresponds to the difference equation model introduced for discrete-time systems in the previous chapter. Such a model is called an *ARX model*, where ARX stands for AutoRegressive with eXogenous input. The results discussed in this section can be extended to ARX models by using the empirical cross-covariance function

$$c_{uy}(l) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N-1} u(i)y(i-l)$$

It follows then that

$$\lim_{N\to\infty} \frac{1}{N} \Phi^T \Phi = \begin{bmatrix} C_{yy} & -C_{uy} \\ -C_{uy} & C_{uu} \end{bmatrix}$$

where the matrices C_{yy} and C_{uy} are defined in the same way as C_{uu}.

The least-squares estimation of an ARX model from measured data is illustrated in Exercise 1.5.

Exercises

Problem 1.1

Consider the sum of squared errors

$$V(\theta) = \sum_{l=0}^{N} e^2(l) = E^T E = (Y - \Phi\theta)^T (Y - \Phi\theta)$$

introduced in (1.3). Show that $V(\theta)$ is minimized by the parameter vector $\theta = \hat{\theta}$ where

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T Y$$

Problem 1.2

a) Show that for the step function $\sigma(k)$

$$(z-1)\sigma(k) = 1 \text{ at } k = -1$$

and

$$(z-1)\sigma(k) = 0 \text{ at } k \geq 0$$

b) If for a given signal $u(k)$ there is at least one polynomial $a(z)$ of order n such that

$$\lim_{N\to\infty} \frac{1}{N} \sum_{l=0}^{N-1} (a(z)u(l))^2 = 0$$

what does this indicate about the order of persistent excitation $u(k)$?

Hint: Theorem 1.2 can be used to solve this problem.

c) Use the polynomial

$$a(z) = z - 1$$

and the results from (a) and (b) to find the greatest possible order of persistent excitation of a step function.

d) Calculate the empirical covariance matrix $C_{uu}(1)$ for the step function. Use this matrix to show that the order of persistent excitation of a step function is 1.

Problem 1.3

a) For the input signal

$$u(k) = \sin \omega kT$$

show that

$$(z^2 - 2z\cos\omega T + 1)u(k) = 0$$

where T is the sampling time.

Hint: Determine $(z^2 - 2z\cos\omega T + 1)u(k)$*, simplify using trigonometric identities.*

b) Find the greatest possible order of persistent excitation of the signal $u(k) = \sin \omega kT$?

c) The autocorrelation function for any signal $x(t)$ is defined as,

$$R_x(\tau) = \lim_{N\to\infty} \frac{1}{N} \sum_{k=0}^{N-1} x(kT)x(kT \pm \tau)$$

It can be shown that for $u(t) = \sin \omega t$,

$$R_u(\tau) = \frac{1}{2}\cos\omega\tau$$

Using these facts show that for the signal $u(k)$

$$C_{uu}(2) = \frac{1}{2}\begin{bmatrix} 1 & \cos\omega T \\ \cos\omega T & 1 \end{bmatrix}$$

Hint: Write the elements of $C_{uu}(2)$ *in terms of the autocorrelation function.*

d) What is the order of persistent excitation of the signal $u(k)$ when

i) $T = \frac{2\pi}{\omega}$ ii) $T \neq \frac{2\pi}{\omega}$

Explain these results.

Problem 1.4 Use the empirical covariance matrices $C_{uu}(1)$, $C_{uu}(2) \ldots C_{uu}(n)$ to show that the order of persistent excitation of sampled white noise is arbitrarily high.

Hint:Use the fact that the correlation between white noise at times T *and* $kT + T$ *is* 0 *and* $\lim_{N\to\infty} \frac{1}{N}\sum_{i=0}^{N-1} u_i^2 = S_0$*, where* S_0 *is the spectral density of the white noise.*

Problem 1.5 Download `cs7_LSsysdat.mat`. The MAT file contains sampled input and output signals of a SISO system, where the input signal is a step (input `u1`, output `y1`), a sinusoid (input `u2`, output `y2`) or white noise (input `u3`, output `y3`). A pulse transfer function is to be determined that approximates the behaviour of the system from which the measurements were taken.

a) From N samples of the inputs and N samples of the output create the measurement matrix Φ for a system of order n. What is the dimension of the matrix Φ produced from these data?

b) Determine the rank of the matrix $\Phi^T\Phi$ for

   i) the sinusoid

   ii) the white noise signal

   Explain the results.

c) From output data generated from white noise, calculate a least squares estimate of the model parameters.

d) i) Estimate the models of order 2, 3 and 4 using the white noise input signal.

   ii) Validate the model for the step and sinusoidal input signals.

   iii) What is the order of the system?

e) Explain the results from (d) with the help of Problems 1.2 and 1.3.b.

Chapter 2

Prediction Error Method

In this chapter we will introduce a framework for identifying linear models of dynamic systems, that includes the ARX model introduced in the previous chapter as a special case: the *Prediction Error Method*. Model structures used in this framework differ in the way information about noise or disturbances is represented and exploited for improving the model. In Chapter 5, this framework is extended to nonlinear model structures; it will be used to train neural networks to learn the behaviour of nonlinear dynamic systems.

Consider the system shown in Figure 2.1, with input signal $u(t)$ and measured output signal $y(t)$. The signal $v(t)$ represents external factors like disturbances or measurement noise that are corrupting the measurement of the output signal, but are not measured. Assume that the input signal $u(t)$ and the output signal $y(t)$ have been sampled with sampling time T, and that sampled data sequences $\{u(0),\ u(T),\ u(2T),\ \ldots,\ u(NT)\}$ and $\{y(0),\ y(T),\ y(2T),\ \ldots,\ y(kT)\}$ are known. Again, for simplicity of presentation we assume $T = 1$ and write $u(k)$ and $y(k)$. The problem we will consider here is: Given the data sequences $\{u(0),\ u(1),\ \ldots,\ u(N)\}$ and $\{y(0),\ y(1),\ \ldots,\ y(N)\}$, find a mathematical model of the system that best fits the data.

The prediction error method solves this problem by minimising the squared sum of *prediction errors*: given the data set

$$Z^N = \{u(k), y(k),\ k = 0, 1, \ldots, N\}$$

and a model structure $\mathcal{M}(\theta)$ (defined below), the task is to find a mapping

$$Z^N \ \rightarrow\ \hat{\theta};$$

that maps the data into a parameter vector $\hat{\theta} \in \mathbb{R}^r$ that best fits the data. "Best fit" here means that the estimated parameter vector minimizes the performance index

$$V_N(\theta, Z^N) = \frac{1}{2N} \sum_{k=p}^{N} \left(y(k) - \hat{y}(k|k-1, \theta)\right)^2, \tag{2.1}$$

where $y(k)$ is the measured output at time k, p is a small number chosen such that the first data vectors are filled with non-zero values (see Chapter 1), and $\hat{y}(k|k-1,\theta)$ is the *one-step-ahead prediction* of the output a time k, predicted by the model with parameters θ. The estimated parameter vector is then

$$\hat{\theta} = \arg \min_{\theta \in \mathcal{D}_{\mathcal{M}}} V_N(\theta, Z^N) \tag{2.2}$$

where $\mathcal{D}_{\mathcal{M}} \subset \mathbb{R}^r$ is a subset of the parameter space to which the estimates are constrained; see below.

Linear Model Structures

First we will have to specify the type of model we want to use.

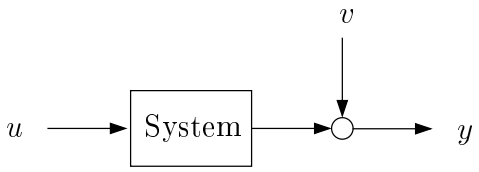

Figure 2.1: Model for system identification

The structure of a discrete-time linear model that represents the system in Figure 2.1 is shown in Figure 2.2. The sampled disturbance signal $v(k)$ is modelled by a white noise sequence $\{e(0),\ e(1),\ \ldots,\ e(k)\}$ which is passed through a filter H. We assume that $e(k)$ is Gaussian distributed and satisfies

$$E[e(k)] = 0, \qquad E[e^2(k)] = \sigma_e^2, \qquad E[e(k)e(k-l)] = 0, \quad l \neq 0$$

The output signal is given by

$$y(k) = G(z^{-1})u(k) + H(z^{-1})e(k) \tag{2.3}$$

where

$$G(z^{-1}) \ = \ z^{-d}\,\frac{b_0 + b_1 z^{-1} + \ldots + b_m z^{-m}}{1 + a_1 z^{-1} + \ldots + a_n z^{-n}} \ = \ z^{-d}\,\frac{B(z^{-1})}{A(z^{-1})}$$

is a transfer function model of the system, and

$$H(z^{-1}) = \frac{1 + c_1 z^{-1} + \ldots + c_l z^{-l}}{1 + d_1 z^{-1} + \ldots + d_n z^{-n}} = \frac{C(z^{-1})}{D(z^{-1})}$$

represents a filter - driven by white noise - that is used to generate a model of the disturbance. We assume that $H(z^{-1})$ and $H^{-1}(z^{-1})$ are stable. The symbol z^{-1} is used to represent both the inverse of the complex variable of the z-transform and the time delay operator, i.e.

$$z^{-1}u(k) = u(k-1), \qquad z^{-d}u(k) = u(k-d) \quad \text{etc.}$$

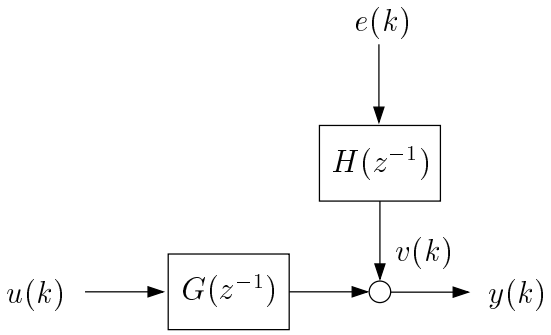

Figure 2.2: Linear model structure

The polynomials $A(z^{-1})$, $C(z^{-1})$ and $D(z^{-1})$ in the above model are *monic*, i.e. we have $A(0) = C(0) = D(0) = 1$. The system is assumed to have a time delay of d sampling periods, and we assume that $d \geq 1$, so that $G(z^{-1})$ is *strictly proper*, i.e. $G(0) = 0$. The disturbance transfer function is *bi-proper*, i.e. $H(0) \neq 0$; because the polynomials C and D are monic we have $H(0) = 1$. This assumption can be made without loss of generality, because the spectral density of the white noise sequence $e(k)$ can be used to express the strength of the disturbance $v(k)$.

Predictor Model

In order to identify a linear model from given data, we will transform the model (2.3) into the form of a *predictor model*. Given input and output data up to time $k-1$, a predictor model provides an estimate of the output signal at time k. We denote the output estimate based on data up to time $k-1$ by $\hat{y}(k|k-1)$, and we define the *prediction error* at time k as

$$\varepsilon(k) = y(k) - \hat{y}(k|k-1).$$

An example of a linear predictor model is the representation (1.10) of a linear model introduced in the previous chapter

$$\hat{y}(k|k-1) = -a_1 y(k-1) - \ldots - a_n y(k-n) + b_0 u(k-d) + \ldots b_m u(k-d-m) \quad (2.4)$$

(here including a time delay d), where an estimate of the output at time k was obtained by using the linear model parameterized by $(a_1 \ldots a_n)$ and $(b_0 \ldots b_m)$; note that the right hand side depends only on past input and output data (up to time $k-1$). In order to construct a predictor model for a general linear model in the form of (2.3), the perturbation term

$$v(k) = H(z^{-1})e(k),$$

which depends on values of e up to time k and is therefore not known at time $k-1$, needs to be replaced by an expression that depends only on past input and output data.

Predicting the Disturbance

The disturbance is modelled by

$$v(k) = H(z^{-1})e(k)$$

or

$$D(z^{-1})v(k) = C(z^{-1})e(k)$$

In time domain we have

$$v(k) + d_1 v(k-1) + \ldots + d_n v(k-n) = e(k) + c_1 e(k-1) + \ldots + c_l e(k-l)$$

or

$$v(k) = e(k) + \text{ function of past values of } e \text{ and } v \tag{2.5}$$

Comparing this with

$$v(k) = H(z^{-1})e(k) - e(k) + e(k) = e(k) + \left(H(z^{-1}) - 1\right) e(k)$$

we see that the term $(H(z^{-1}) - 1)e(k)$ is equal to the past data term in (2.5). Because $E[e(k)] = 0$, the best estimate of $v(k)$ at time k is therefore

$$\hat{v}(k|k-1) = (H(z^{-1}) - 1)e(k) \tag{2.6}$$

provided the right hand side can be computed from past data. The white noise sequence driving the disturbance model $H(z^{-1})$ is unknown, but it can be expressed in terms of the disturbance $v(k)$ by inverting the filter transfer function

$$e(k) = H^{-1}(z^{-1})v(k)$$

Substituting in (2.6) yields

$$\hat{v}(k|k-1) = \left(1 - H^{-1}(z^{-1})\right) v(k) \tag{2.7}$$

Given input and output data, the disturbance can be computed from

$$v(k) = y(k) - G(z^{-1})u(k) \tag{2.8}$$

Since the right hand side of (2.7) depends only on values of v up to time $k-1$, it can be computed from input and output data up to time $k-1$.

Example 2.1

Consider a disturbance model

$$H(z^{-1}) = \frac{1}{1 - az^{-1}}$$

where $|a| < 1$. From $v(k) = H(z^{-1})e(k)$ we have

$$v(k) = e(k) + av(k-1)$$

thus the best estimate of $v(k)$ is

$$\hat{v}(k|k-1) = av(k-1)$$

which is just the estimate provided by (2.7)

$$\hat{v}(k|k-1) = (1 - 1 + az^{-1})v(k)$$

Predicting the Output

At time k, the best estimate of the output in (2.3) that takes the disturbance into account is

$$\hat{y}(k|k-1) = G(z^{-1})u(k) + \hat{v}(k|k-1)$$

Substituting from (2.7) gives

$$\hat{y}(k|k-1) = G(z^{-1})u(k) + \left(1 - H^{-1}(z^{-1})\right) v(k)$$

The right hand side involves only values of v up to $k-1$; these can be computed from (2.8), yielding

$$\hat{y}(k|k-1) = G(z^{-1})u(k) + \left(1 - H^{-1}(z^{-1})\right) \left(y(k) - G(z^{-1})u(k)\right)$$

or

$$\hat{y}(k|k-1) = H^{-1}(z^{-1})G(z^{-1})u(k) + \left(1 - H^{-1}(z^{-1})\right) y(k) \tag{2.9}$$

Equation (2.9) is called the *predictor form* of the model (2.3); it is shown in Figure 2.3. Many system identification techniques - those using the *prediction error method* - are based on this type of model. Note that the right hand side of (2.9) involves only past input and output data, i.e. the predictor model is strictly proper.

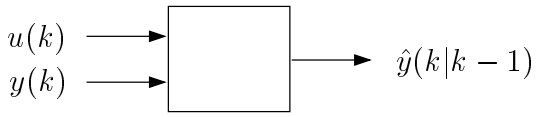

Figure 2.3: Predictor model

Model Structures

Equation (2.9) represents the predictor form of the linear model given by (2.3). When estimating a mathematical model of a system from experimental data, it is useful to

distinguish between a model structure and a particular model. A *model structure* is a set of models that is suitably parameterized, and the task of system identification is to find the best model within that set, i.e. to find the numerical values of the parameter variables that best fit the data, in the sense of a performance index to be specified. A particular *model* is generated when numerical values are substituted for the parameter variables.

The model structure associated with equation (2.3) is the set

$$\mathcal{M} = \{G(z^{-1}, \theta), H(z^{-1}, \theta) \mid \theta \in \mathcal{D}_{\mathcal{M}} \subset \mathbb{R}^r\} \tag{2.10}$$

of discrete-time transfer function pairs (G, H) with parameter variables $\theta_1, \ldots, \theta_r$, which have been collected in a parameter vector θ. The parameter vector is constrained to be in a set

$$\mathcal{D}_{\mathcal{M}} = \{\theta \in \mathbb{R}^r \mid H^{-1} \text{ and } H^{-1}G \text{ stable}, \ G(0, \theta) = 0, \ H(0, \theta) = 1\} \tag{2.11}$$

These constraints ensure that the predictor model (2.9) is stable and depends only on past data. Note that defining a model structure implies a decision on the order of the polynomials of the transfer functions G and H.

We will sometimes use the concept of a *true model*, by which we mean a linear model with transfer functions $G_0(z^{-1})$ and $H_0(z^{-1})$. We will then assume that the output data have been generated using this model and the white noise sequence $e_0(k)$.

Process Innovation

From the predictor form (2.9) we obtain

$$y(k) - \hat{y}(k|k-1) = -H^{-1}(z^{-1}, \theta)G(z^{-1}, \theta)u(k) + H^{-1}(z^{-1}, \theta)y(k)$$

or

$$\varepsilon(k, \theta) = H^{-1}(z^{-1}, \theta)\Big(y(k) - G(z^{-1}, \theta)u(k)\Big)$$

If the true model $(G_0(z^{-1}), H_0(z^{-1}))$ is substituted above and $(G_0, H_0) \in \mathcal{M}$, we have

$$\varepsilon(k, \theta_0) = H_0^{-1}(z^{-1})v_0(k) = e_0(k) \tag{2.12}$$

which shows that the prediction error becomes white noise; in other words, the prediction error at time k is uncorrelated to past data. The term $e_0(k)$ is called the *innovation at time k*, it represents the part of the output $y(k)$ that cannot be predicted from past data. On the other hand, if G and H do not represent the true model, the prediction error will not be white. The aim of parameter estimation is to reduce the prediction error to white noise by extracting all available information from the data - if the prediction error is not white there is still correlation to past data, i.e. information that can be extracted and used to improve the model.

ARX and ARMAX Models

The linear model structure (2.3) is quite general and allows a variety of model structures as special cases, corresponding to different assumptions about the disturbance model H. The simplest (and widely used) model structure is that of the ARX model introduced in the previous chapter. The transfer functions in this model structure are

$$G(z^{-1},\theta) = z^{-d}\,\frac{B(z^{-1},\theta)}{A(z^{-1},\theta)}, \qquad H(z^{-1},\theta) = \frac{1}{A(z^{-1},\theta)} \tag{2.13}$$

Substituting in the predictor model (2.9) yields

$$\hat{y}(k|k-1,\theta) = z^{-d}Bu(k) + (1-A)y(k) \tag{2.14}$$

(dropping arguments for notational simplicity), or

$$\hat{y}(k|k-1,\theta) = b_0u(k-d) + \ldots + b_mu(k-d-m) - a_1y(k-1) - \ldots - a_ny(k-n)$$

Note that the predictor form (2.14) of the ARX model does not have any poles. Stability of the predictor is therefore guaranteed without further assumptions. Moreover, the model assumptions $G(0) = 0$ and $H(0) = 1$ are built into this model structure by assuming that A and C are monic, and that $d \geq 1$. Therefore we have $\mathcal{D}_{\mathcal{M}} = \mathbb{R}^r$.

In the previous chapter it was shown how an ARX model can be expressed in the form of a linear regressor model

$$\hat{y}(k|k-1,\theta) = \varphi^T(k)\theta$$

by defining the parameter vector

$$\theta^T = [-a_1 \ \ldots \ -a_n \ \ b_0 \ \ldots \ b_m]$$

and the regressor vector at time k

$$\varphi^T(k) = [y(k-1) \ \ldots \ y(k-n) \ \ u(k-d) \ \ldots \ u(k-d-m)]$$

The regressor form can be used to compute an estimate $\hat{\theta}$ that minimizes the sum of squared prediction errors

$$\sum_{k=p}^{N} \varepsilon^2(k,\theta), \quad \text{where} \quad \varepsilon(k,\theta) = y(k) - \hat{y}(k|k-1,\theta)$$

where $p = \max(n, d+m)$ is the number of initial samples required for filling up the regressor vector. The minimum can be found by solving a linear, unconstrained least squares problem. Because of its simplicity, this model structure is widely used; however the price for this simplicity is a lack of flexibility on the noise model.

A model structure that allows to incorporate more information about the perturbation signal $v(k)$ is the ARMAX model structure. ARMAX stands for ARX together with a

Moving Average of the prediction error, see equation (2.16) below. In an ARMAX model the plant model G is the same as in an ARX model, but the disturbance model is

$$H(z^{-1},\theta) = \frac{C(z^{-1},\theta)}{A(z^{-1},\theta)}$$

The numerator polynomial C allows to include estimated noise properties in the model. The predictor form of the ARMAX model is

$$\hat{y}(k|k-1,\theta) = z^{-d}\,\frac{B}{C}u(k) + \left(1 - \frac{A}{C}\right)y(k) \tag{2.15}$$

To bring this into the form of a difference equation, we multiply by C to get

$$C\hat{y}(k|k-1,\theta) = z^{-d}Bu(k) + (C-A)y(k)$$

or

$$\hat{y}(k|k-1,\theta) = z^{-d}Bu(k) + (1-A)y(k) + (C-1)\varepsilon(k)$$

where $\varepsilon(k)$ is the prediction error, and we used fact that A and C are monic. We thus have

$$\begin{aligned}\hat{y}(k|k-1,\theta) = {} & b_0u(k-d) + \ldots + b_mu(k-d-m) \\ & - a_1y(k-1) - \ldots - a_ny(k-n) + c_1\varepsilon(k-1) + \ldots + c_l\varepsilon(k-l)\end{aligned} \tag{2.16}$$

Note that compared with the ARX model, the right hand side contains a moving average of past prediction errors. The above model can be expressed in regressor form by defining

$$\theta^T = [-a_1 \ \ldots \ -a_n \ \ b_0 \ \ldots \ b_m \ \ c_1 \ \ldots \ c_l]$$

and

$$\varphi^T(k,\theta) = [y(k-1) \ \ldots \ y(k-n) \ \ u(k-d) \ \ldots \ u(k-d-m) \ \ \varepsilon(k-1) \ \ldots \ \varepsilon(k-l)]$$

The regressor vector is now a function of the parameter vector θ, because it contains past prediction errors. The regressor model is

$$\hat{y}(k|k-1,\theta) = \varphi^T(k,\theta)\theta$$

which looks like a linear regressor model but is in fact nonlinear due to the dependence of φ on θ. This form is referred to as *pseudolinear regressor form*. Unlike the linear regressor form of the ARX model, it requires an iterative search for the parameter estimate $\hat{\theta}$.

In contrast to ARX models, the predictor form of an ARMAX model has poles - the roots of the disturbance polynomial C. To guarantee stability of the predictor, we need to impose the constraint that the polynomial $C(z^{-1})$ has its roots - the values of z^{-1} for which $C(z^{-1}) = 0$ - outside the unit disc (which implies that the inverse roots z are inside the unit disc). We will say that C is stable if it satisfies this condition, and for ARMAX models we define $\mathcal{D}_{\mathcal{M}}$ as the set of parameter vectors θ such that C is stable.

Computing the Estimate

For ARX models, the numerical solution of the minimization problem in (2.2) was discussed in Chapter 1 and is briefly reviewed here. We introduce the vectors

$$Y = \begin{bmatrix} y(p) \\ \vdots \\ y(N) \end{bmatrix}, \qquad \hat{Y} = \begin{bmatrix} \hat{y}(p|p-1,\theta) \\ \vdots \\ \hat{y}(N|N-1,\theta) \end{bmatrix}$$

Using the regressor form of ARX models, we can then write

$$\hat{Y} = \Phi\theta, \qquad \Phi = \begin{bmatrix} \varphi^T(p) \\ \vdots \\ \varphi^T(N) \end{bmatrix}$$

The estimate of the parameter vector can therefore be obtained by solving

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{2N} E(\theta)^T E(\theta),$$

where $E(\theta) = Y - \Phi\theta$ is the vector of prediction errors. The solution is provided by the *normal equation*

$$\Phi^T\Phi\theta = \Phi^T Y$$

which can be solved for

$$\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y$$

provided the inverse exists. The existence of the inverse depends on the input signal $u(k)$ being *persistently exciting* of sufficient order. In practice, computing the inverse is usually avoided and a square root algorithm is used instead.

For ARMAX models, the fact that the regressor vector $\varphi(k,\theta)$ depends on the parameters prevents such a straightforward solution. An iterative search for the minimizer of $V_N(\theta)$ is required, based on the gradient of the predicted output with respect to the parameters. Iterative search techniques will be discussed later in the context of neural network training. Here we will only observe that the gradient vector can be obtained in a straightforward way from the regressor vector. Define the gradient vector $\psi(k,\theta) \in \mathbb{R}^r$ as

$$\psi(k,\theta) \;=\; \frac{\partial}{\partial\theta}\hat{y}(k|k-1,\theta) \;=\; \left[\frac{\partial\hat{y}_k}{\partial a_1} \;\cdots\; \frac{\partial\hat{y}_k}{\partial a_n} \;\; \frac{\partial\hat{y}_k}{\partial b_0} \;\cdots\; \frac{\partial\hat{y}_k}{\partial b_m} \;\; \frac{\partial\hat{y}_k}{\partial c_1} \;\cdots\; \frac{\partial\hat{y}_k}{\partial c_l}\right]^T$$

where the simplified notation $\hat{y}_k$ is used for $\hat{y}(k|k-1,\theta)$. From (2.16) we have

$$C(z^{-1},\theta)\frac{\partial}{\partial a_i}\,\hat{y}(k|k-1,\theta) = -y(k-i)$$

$$C(z^{-1},\theta)\frac{\partial}{\partial b_i}\,\hat{y}(k|k-1,\theta) = u(k-d-i)$$

and

$$\hat{y}(k-i|k-i-1,\theta) + C(z^{-1},\theta)\frac{\partial}{\partial c_i}\,\hat{y}(k|k-1,\theta) = y(k-i)$$

thus

$$C(z^{-1},\theta)\frac{\partial}{\partial c_i}\,\hat{y}(k|k-1,\theta) = \varepsilon(k-i)$$

Substituting the above in $\psi(k,\theta)$ shows that

$$C(z^{-1},\theta)\psi(k,\theta) = \varphi(k,\theta)$$

In other words, the gradient vector can be obtained by passing the regressor vector through a filter

$$\psi(k,\theta) = \frac{1}{C(z^{-1},\theta)}\,\varphi(k,\theta). \tag{2.17}$$

Neural Network Based Nonlinear Model Structures

The prediction error method can be applied to nonlinear model structures - in this course it will be used for the training of neural network based models. We will consider models in a nonlinear regressor form

$$y(k) = g(\varphi(k),\theta) + e(k) \tag{2.18}$$

where θ is a vector containing the model parameters, $\varphi(k)$ is a regressor vector that can be constructed from past data, $e(k)$ represents a white noise sequence, and g is a static nonlinear function. Note that in (2.18) we assume that the perturbation $v(k)$ shown in Figure 2.1 is again modelled as filtered white noise, and that it has already been brought into the form

$$v(k) = e(k) + \text{ past values of } v$$

The predictor form of this model is then

$$\hat{y}(k|k-1) = g(\varphi(k),\theta)$$

Figure 2.4 shows how a neural network can be used to represent this predictor model. The parameter vector θ contains the neural network parameters which can be tuned to obtain a good approximation of the nonlinear function g.

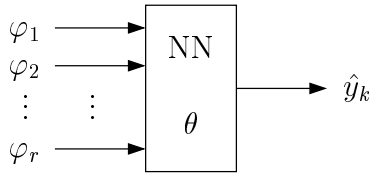

Figure 2.4: A neural network as predictor model

In analogy to the linear ARX and ARMAX model structures, we will use structures with similar assumptions on the noise model for neural network based predictors. A neural network based ARX structure - referred to as NNARX structure - is shown in Figure 2.5. A structure similar to the linear ARMAX structure - referred to as NNARMAX structure - is shown in Figure 2.6. Note that because of the feedback involved in this structure, stability of the predictor is not guaranteed when the data are presented sequentially to the network. Stability issues for nonlinear predictors are usually treated in a heuristic fashion. In contrast, the NNARX predictor involves only a static feedforward network and is guaranteed to be stable.

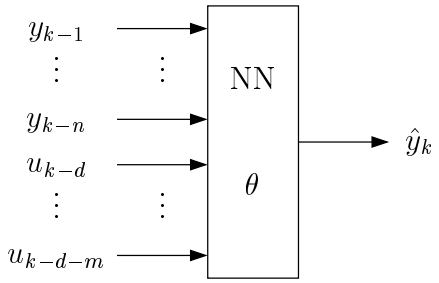

Figure 2.5: NNARX model structure

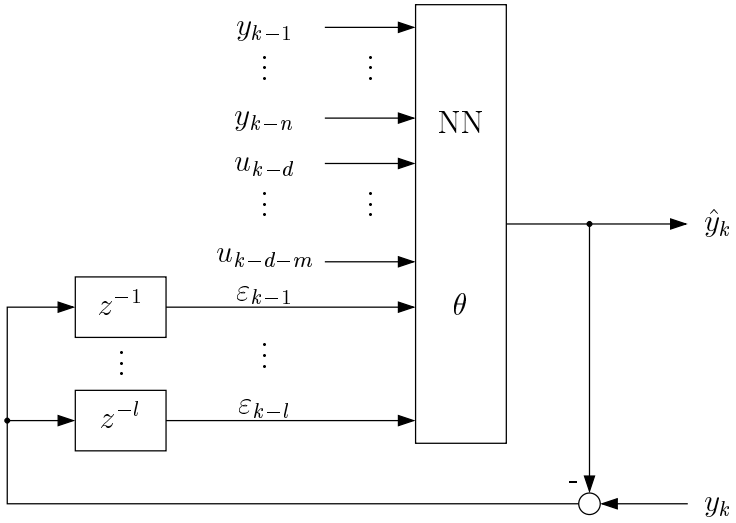

Figure 2.6: NNARMAX model structure

Exercises

Problem 2.1

From Stud.IP the file *armax_ident.mat* can be downloaded. It contains vectors u and y of sampled input and output data, respectively, and a continuous-time transfer function G that represents the true plant model. The data have been sampled with a sampling interval $T = 0.3$. The output data are corrupted by filtered noise.

Use the MATLAB Identification Toolbox to estimate a linear model that fits the data.

a) Estimate an ARMAX model. Start by estimating first order plant and disturbance models. When estimating, use the *simulation* option for $focus$ in the *polynomial models* window (this option uses a variant of ARMAX estimation that puts more emphasis on lower frequencies). Use the *LTI viewer* to display the step response of the estimated model, and compare it with the step response of the true plant model G. Increase the order of plant or disturbance model, if necessary, to obtain a better fit.

b) Try to estimate an ARX model and compare it with the ARMAX model obtained in (a).

Chapter 3

Multilayer Perception Networks

In this course we will study the use of neural networks for modelling and control of nonlinear systems. A common application of neural networks is the solution of classification problems (e.g. pattern recognition). In control applications, on the other hand, neural networks are mainly used to approximate nonlinear functions. Consider a continuous, nonlinear function $g : \mathbb{R}^r \to \mathbb{R}$

$$y = g(\varphi_1, \varphi_2, \ldots, \varphi_r)$$

as illustrated in Figure 3.1 for the case $r = 2$.

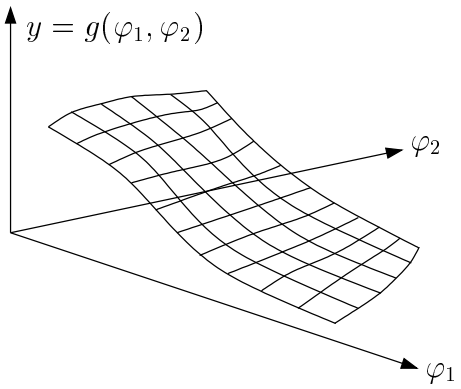

Figure 3.1: Nonlinear function $y = g(\varphi_1, \varphi_2)$

Assume that at a number of points in the r-dimensional input space the values of g are known. A problem we will consider in this course is that of finding an approximation of g that is as close as possible to the known values at the given points.

Figure 3.2 indicates how a neural network can be used to obtain such an approximation. The neural network - the block labelled NN - represents a static nonlinear function of the inputs $\varphi_1, \ldots, \varphi_r$. This function itself depends on a number of parameters which can be tuned. The network output $\hat{y}$ is taken to be an approximation of the value of

$g(\varphi_1, \ldots, \varphi_r)$. When the values of the given points in input space are presented to the network one by one, the network output $\hat{y}$ is compared with the known value of g at each point, and the error ε is used by a *learning algorithm* to tune the parameters of the network such the best possible fit (in a sense to be specified) is obtained. This process is referred to as *training* of a neural network.

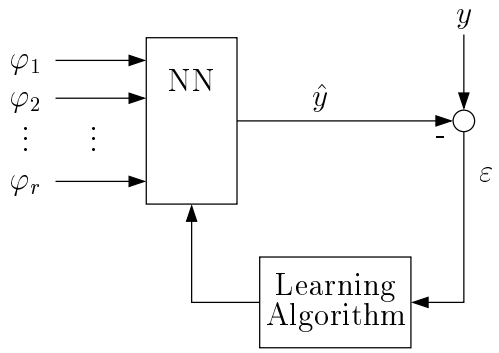

Figure 3.2: Nonlinear approximation using a neural network

Nonlinear Plant Models for Control

In the previous chapter we encountered the ARX model structure

$$\hat{y}(k) = -a_1 y(k-1) - \ldots - a_n y(k-n) + b_1 u(k-1) + \ldots + b_m u(k-m)$$

In this course we will consider nonlinear model structures like

$$\hat{y}(k) = g\left(y(k-1), \ldots, y(k-n), u(k-1), \ldots, u(k-m)\right)$$

where $g : \mathbb{R}^{n+m} \to \mathbb{R}$ is a continuous, nonlinear function. This model structure can be interpreted as a nonlinear version of the linear ARX structure. If a sequence of input and output data for a given nonlinear system is available, one can at time k apply past input and output data as inputs to the neural network in Figure 3.2, compare the network output with the actual value of $y(k)$ and adjust the network parameters accordingly. If this process is repeated until a satisfactory approximation is obtained, we say that the network has been trained to *learn* the nonlinear behaviour of the system that generated the data.

Once a neural network has been trained, there are several ways of using it for designing a controller. One possibility is shown in Figure 3.3. The control loop contains the nonlinear plant and a digital, sampled-data controller. At each sampling instant, input and output samples are applied to a neural network that has been trained on the nonlinear dynamic behaviour of the plant, and a linearized model is extracted from the network. This procedure is called *instantaneous linearization*. The parameters of the linearized model are passed to a controller adjustment module that determines the values of the controller

parameters for the given model, and updates the controller at each sampling instant. Control schemes that can be implemented in this way include pole placement design, minimum variance control and predictive control.

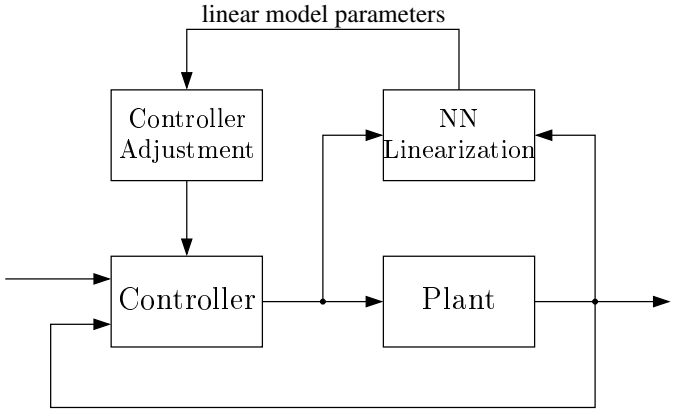

Figure 3.3: Neural network based control using instantaneous linearization

Multilayer Perceptron Networks

The development of artificial neural networks (ANNs) was originally motivated by research on the human brain. The human brain contains a network of about 10^{11} neurons or nerve cells, which are heavily interconnected. Although ANNs are sometimes considered to be simplified models of the human brain, this interpretation is somewhat misleading since the function of the brain is very complex and still not well understood. Nevertheless, a simplified view of biological neurons - illustrated in Figure 3.4 - has inspired the construction of artificial neurons.

In this simplified view, a neuron has four principal components:

- The *dendrites* represent a highly branching tree of fibres that carry electrical signals to the cell body. There are typically 10^3 to 10^4 dendrites per neuron. The dendrites can be interpreted as the input channels of a neuron.
- The *soma* or cell body realizes the logical functions of the neuron. The soma contains the nucleus and the protein synthesis machinery of the nerve cell.
- The *axon* is a single long nerve fibre attached to the soma that serves as the output channel of the neuron. Signals are generally converted into pulse sequences and propagated along the axon to target cells (i.e. other neurons).
- A *synapse* is a point of contact between an axon of one cell and a dendrite of another cell. The strength of a synaptic connection is variable and may increase if frequently

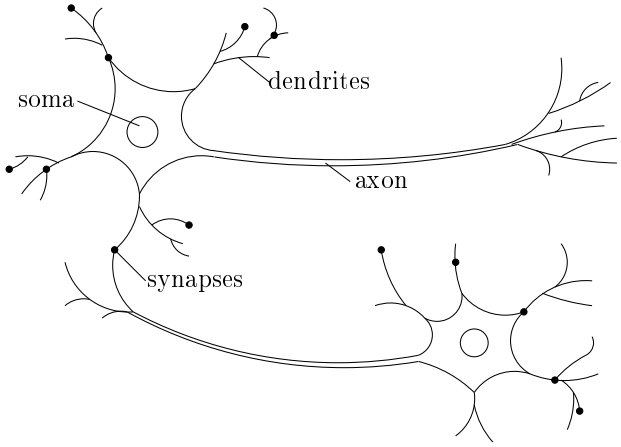

Figure 3.4: Biological neurons

stimulated. The storage of information in a neuron is thought to be concentrated in the pattern and strength of its synaptic connections.

Artificial Neurons

A basic model of an artificial neuron is shown in Figure 3.5. The neuron is modelled as a multi-input nonlinear device with r inputs $\varphi_1,\ \varphi_2, \ldots,\ \varphi_r$, one output v and weighted interconnections w_i, representing synaptic weights. Moreover, an extra input with a value fixed to 1 is provided that can be used to generate a *bias* w_0.

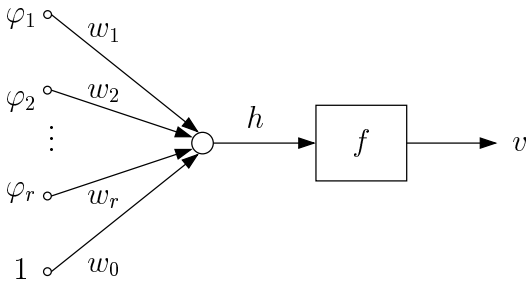

Figure 3.5: Artificial Neuron

In this model of an artificial neuron, the sum h of the r weighted inputs and the bias is passed through a static nonlinear function $f(h)$ according to

$$v = f(h) = f\left(\sum_{i=1}^{r} w_i \varphi_i + w_0\right)$$

Introducing the column vectors

$$w = \begin{bmatrix} w_1 \\ \vdots \\ w_r \end{bmatrix}, \qquad \varphi = \begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_r \end{bmatrix}$$

this can be written as

$$v = f(w^T \varphi + w_0)$$

The nonlinear function f is called the *activation function* of the neuron. Three types of activation function are commonly used: step functions, linear functions and sigmoid functions.

Step Activation Function

The step function as activation function is defined by

$$v(h) = \sigma(h) = \begin{cases} 1, & h \geq 0, \\ 0, & h < 0. \end{cases}$$

To illustrate the effect of weights and bias on a neuron, consider the single-input neuron in Figure 3.6.

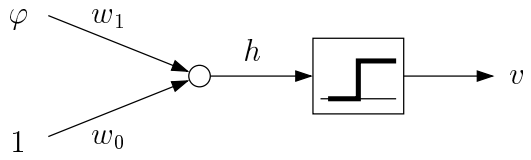

Figure 3.6: Single-input step function neuron

The output is

$$v = \sigma(w_1 \varphi + w_0)$$

and the values of v as functions of h and φ are shown in Figure 3.7.

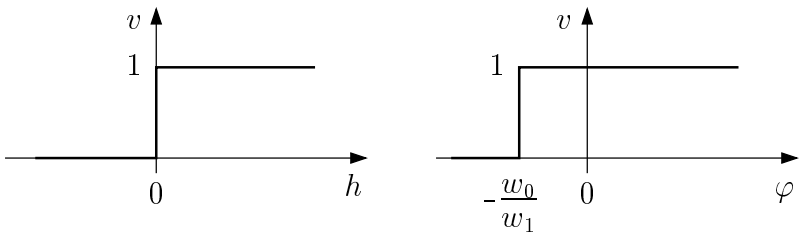

Figure 3.7: Response of a step function neuron

Linear Activation Function

The output of a linear activation function is equal to its input

$$v(h) = h$$

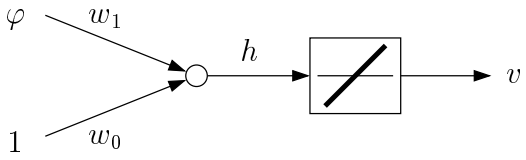

Figure 3.8: Single-input linear function neuron

For a single-input neuron with linear activation function, the output v as a function of h and φ, respectively, is shown in Figure 3.9.

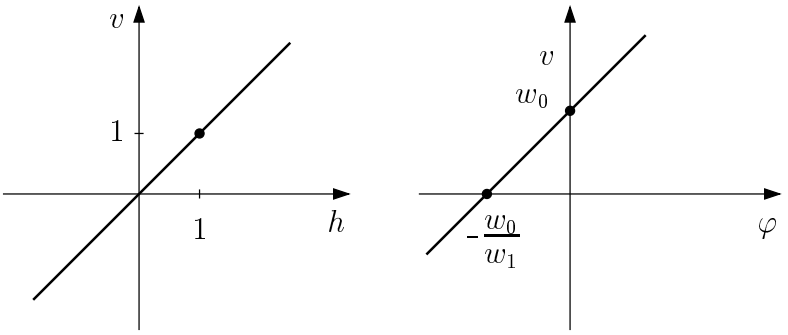

Figure 3.9: Response of a linear function neuron

Sigmoid Activation Function

There are two types of sigmoid activation functions: the *log-sigmoid* function and the *tanh-sigmoid* function.

The log-sigmoid activation function is

$$v(h) = f(h) = \frac{1}{1 + e^{-h}}$$

For a single-input neuron with log-sigmoid activation function, the output v as a function of h and φ, respectively, is shown in Figure 3.11. Note that while the ratio w_0/w_1 determines the location of the inflection point of the function $v(\varphi)$, the value of the weight w_1 determines the slope at the inflection point.

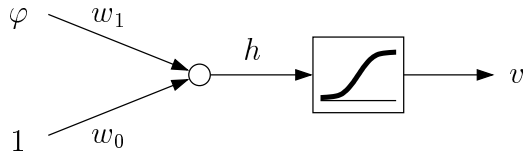

Figure 3.10: Single-input log-sigmoid function neuron

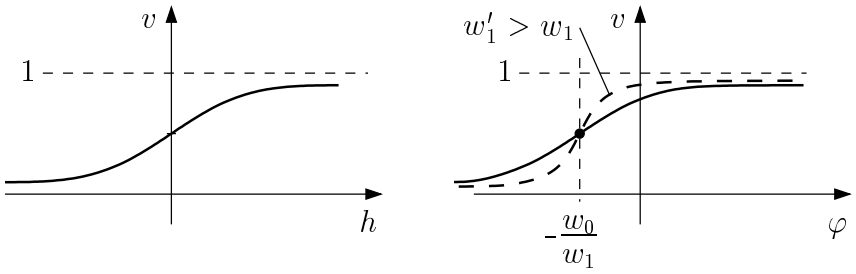

Figure 3.11: Response of a log-sigmoid function neuron

The tanh-sigmoid activation function is

$$v = f(h) = \frac{e^h - e^{-h}}{e^h + e^{-h}}$$

For a single-input neuron with tanh-sigmoid activation function, the output v as a function of h and φ, respectively, is shown in Figure 3.13.

Artificial Neural Networks

Having defined a basic neuron model, we can now turn to networks formed by connecting single neurons. A single-layer network containing s neurons is shown in Figure 3.14. The network has r inputs $\varphi_1, \ldots, \varphi_r$, a bias and s outputs $v_1, \ldots, v_s$. All neurons are assumed to have the same activation function f.

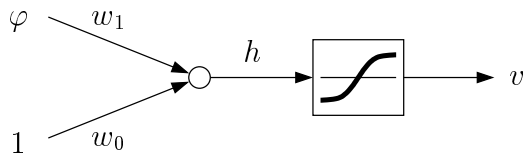

Figure 3.12: Single-input tanh-sigmoid function neuron

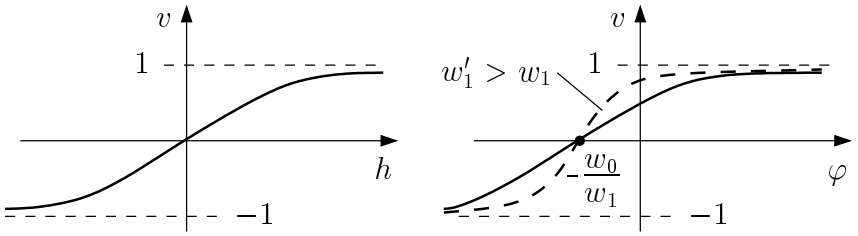

Figure 3.13: Response of a tanh-sigmoid function neuron

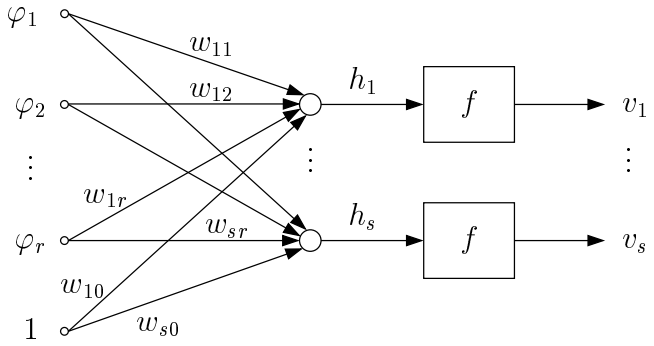

Figure 3.14: Single layer network

The output of the summing junction of the i^{th} neuron is

$$h_i = [w_{i1} \;\; w_{i2} \;\; \ldots \;\; w_{ir}] \begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_r \end{bmatrix} + w_{i0}$$

where w_{ij} is the gain from input j to the i^{th} neuron. Defining the weight vector

$$w_i^T = [w_{i1} \;\; w_{i2} \;\; \ldots \;\; w_{ir}]$$

we have

$$\begin{bmatrix} h_1 \\ \vdots \\ h_s \end{bmatrix} = \begin{bmatrix} w_1^T \\ \vdots \\ w_s^T \end{bmatrix} \varphi + \begin{bmatrix} w_{10} \\ \vdots \\ w_{s0} \end{bmatrix}$$

Introducing the weight matrix W, the bias vector w_0 and the vector of summing junction outputs h, this can be written in a more compact form as

$$h = W\varphi + w_0$$

The vector v of network outputs is then

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_s \end{bmatrix} = f(W\varphi + w_0)$$

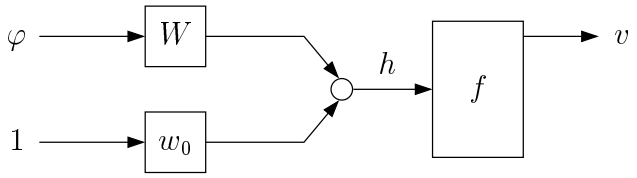

Figure 3.15: Perceptron layer

A single-layer network of this form is called a *perceptron network*. A block diagram representation of a perceptron network is shown in Figure 3.15.

Multilayer Perceptron Network

Several perceptron layers can be connected in series to form a *multilayer perceptron (MLP) network*. As an example, a network with two layers of neurons - three neurons in the first layer and two in the second layer - is shown in Figure 3.16. This network has r inputs and two outputs, and a bias in each layer. In a multilayer network, the last layer - the one that generates the network outputs - is called the *output layer*, whereas the other layers are called *hidden layers*, because they are hidden between inputs and output layer.

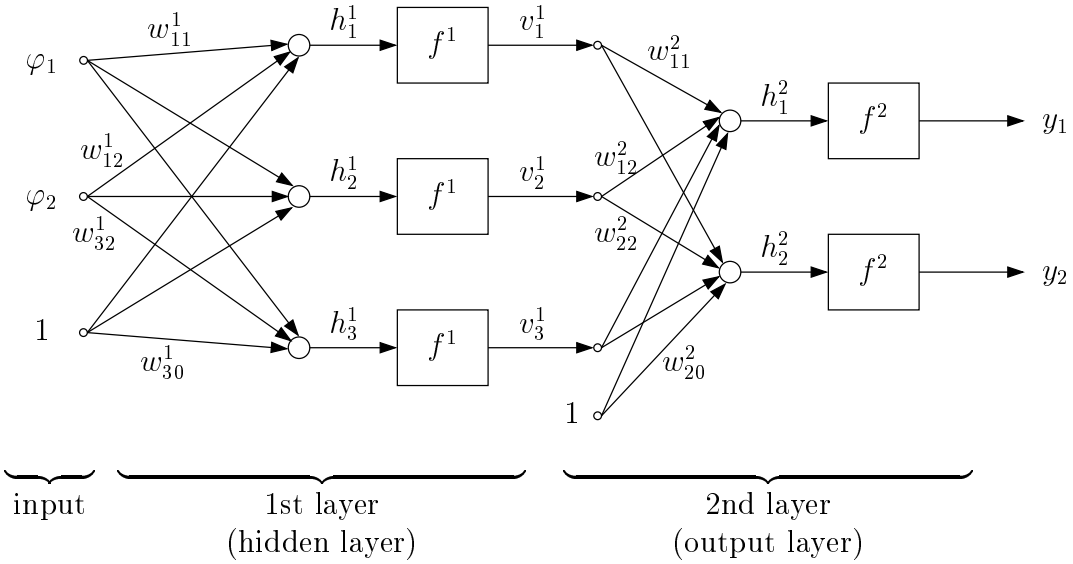

Figure 3.16: Multilayer perceptron network

Here the superscript i is used to denote variables or parameters of the i^{th} layer. At the output of the first layer we have

$$v^1 = \begin{bmatrix} v_1^1 \\ v_2^1 \\ v_3^1 \end{bmatrix} = f^1(W^1\varphi + w_0^1)$$

The network output - the output of the second layer - is

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f^2(W^2 v^1 + w_0^2)$$

or in general

$$y = f^2\left(W^2 f^1(W^1 \varphi + w_0^1) + w_0^2\right) \tag{3.1}$$

A block diagram representation of a two-layer perceptron network is shown in Figure 3.17.

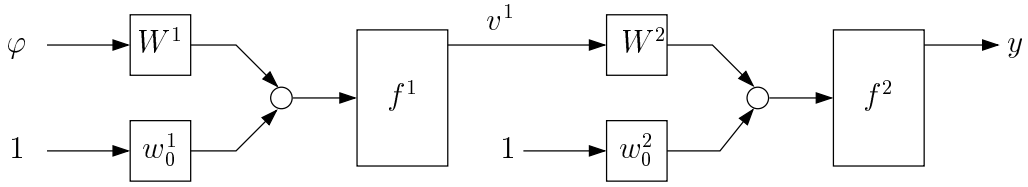

Figure 3.17: Multilayer perceptron network - block representation

Universal Approximation

For control applications, a commonly used network structure is a two-layer perceptron network with sigmoidal activation functions in the hidden layer, and linear activation functions in the output layer. An important property of such networks is their *universal approximation capability*: Any given real continuous function $g : \mathbb{R}^r \to \mathbb{R}$ can be approximated to any desired accuracy by a two-layer *sig-lin* network (here assuming that the network has only a single output). The proof of this result gives however no indication about the number of hidden units required for achieving the desired accuracy; this issue will be explored in two exercises.

Exercises

Problem 3.1

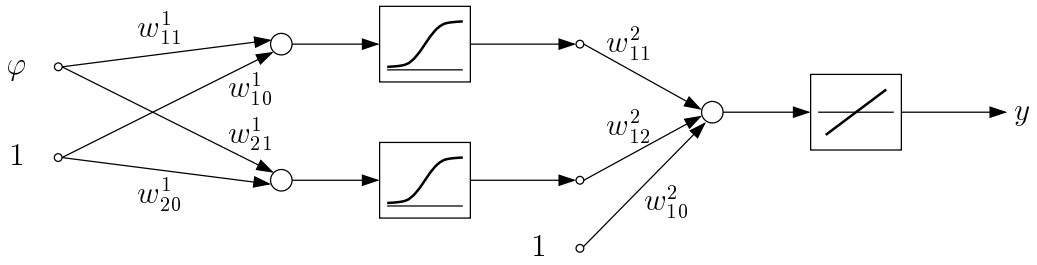

Figure 3.18: Single-input MLP

Consider the single-input MLP network in Figure 3.18. Find the weights W^1, W^2 and bias values w_0^1, w_0^2 required to approximate the function shown in Figure 3.19

a) by inspecting the figure and verify by using the *nntool* of MATLAB,

b) by training the neural network using the *Fitting app* of MATLAB.

Hint: Download the MATLAB data file *approx.mat* from Stud.IP, which contains the input vector *phi* and the target vector *g_phi* corresponding to Figure 3.19. See solutions for more details.

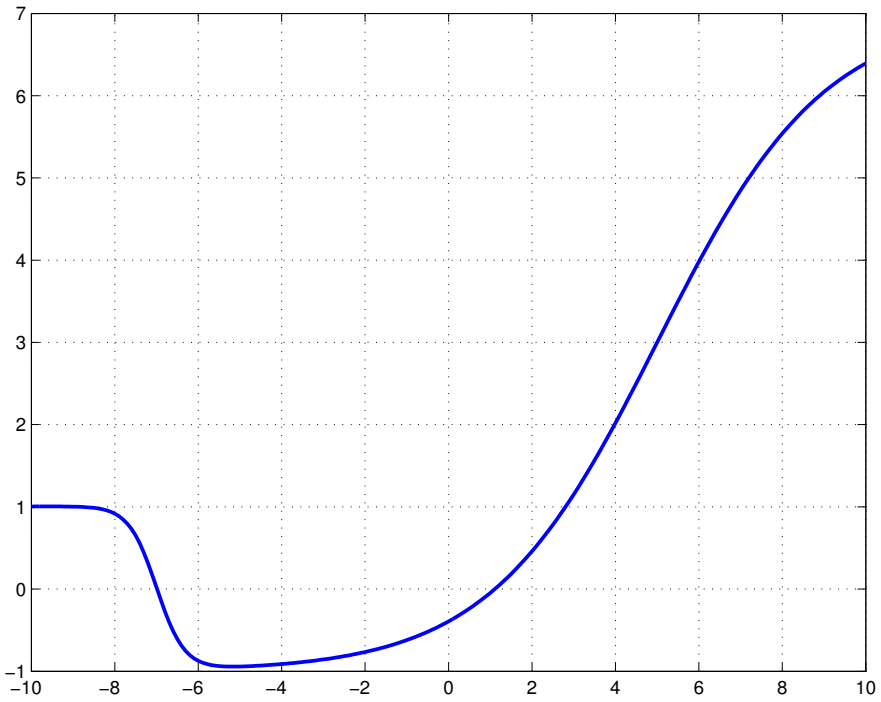

Figure 3.19: Nonlinear function

Problem 3.2

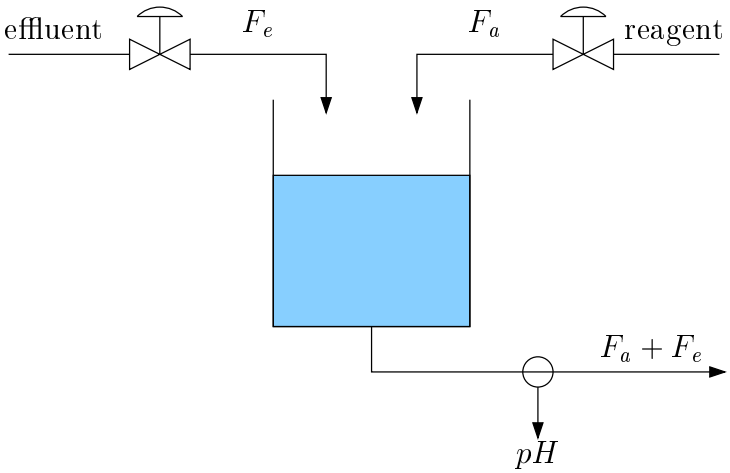

Figure 3.20: CSTR pH process

A nonlinear Simulink model of a continuous stirred tank reactor (CSTR) pH process can be downloaded from the web site for this exercise. The CSTR has two input streams: the *effluent stream* (with flow rate F_e), and the *reagent stream* (with flow rate F_a), see Figure 3.20. The effluent is assumed to be a base and the reagent to be an acid. The reagent stream is used to control the pH value at the outlet. A block diagram of the process

is shown in Figure 3.21, where the control input is the reagent flow rate, the controlled output is the pH value at the outlet, and possible changes in the reagent flow rate are modelled as disturbance.

Tune the PID controller in the control loop provided in the Simulink model such that the system tracks the given reference step changes with settling time and overshoot as small as possible.

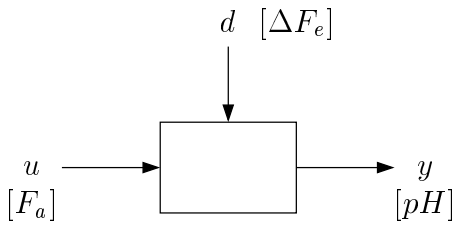

Figure 3.21: Block diagram

Chapter 4

Solving the Estimation Problem

In this chapter we will discuss how multilayer perceptron networks can be used to identify and model the dynamic behaviour of nonlinear systems. We will first review some basic concepts about nonlinear optimization; these will then be applied to the training of neural networks. We will finally see how multilayer perceptron networks having the nonlinear NNARX and NNARMAX model structures discussed in Section 2 can be trained.

Performance Learning

The process of adjusting the weights and bias values in a multilayer perceptron network like that shown in Figure 3.16 such that a good approximation of a nonlinear function $y = g(\varphi)$ is achieved, is called *training* the network, or - seen from the network - as *learning*. If the adjustment is carried out such that a performance index - a measure of the accuracy of the approximation - is minimized, this process is called *performance learning*. We will use neural networks for modelling nonlinear dynamic systems, and the performance index we are interested in is the sum of squared prediction errors (2.1)

$$V(\theta, Z^N) = \frac{1}{2N} \sum_{k=p}^{N} \left(y(k) - \hat{y}(k|k-1,\theta)\right)^2 = \frac{1}{2N} \sum_{k=p}^{N} \varepsilon^2(k,\theta) \tag{4.1}$$

where $Z^N = \{y(k), \varphi(k); k = 1, \ldots, N\}$ represents a set of N samples of measured input and output data of the system to be modelled, $y(k)$ is the measured output at sampling instant k, $\hat{y}(k)$ the output predicted by the network, and the parameter vector $\theta \in \mathbb{R}^{n_p}$ contains all adjustable network parameters, i.e. weights and bias values. The objective is then to find the minimizing value

$$\hat{\theta} = \arg\min_{\theta} V(\theta, Z^N)$$

In the neighborhood of a given value θ_0 of the parameter vector, the performance index can be developed into a Taylor series

$$V(\theta) = V(\theta^0) + \nabla V^T(\theta)\Big|_{\theta^0} (\theta - \theta^0) + \frac{1}{2} (\theta - \theta^0)^T \nabla^2 V(\theta)\Big|_{\theta^0} (\theta - \theta^0) + \ldots$$

(to simplify notation, the argument Z^N has been dropped), where

$$\nabla V(\theta) = \left[\frac{\partial V}{\partial \theta_1} \cdots \frac{\partial V}{\partial \theta_1}\right]^T$$

denotes the gradient of $V(\theta)$ and

$$\nabla^2 V(\theta) = \begin{bmatrix} \frac{\partial^2 V}{\partial \theta_1^2} & \frac{\partial^2 V}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 V}{\partial \theta_1 \partial \theta_{n_p}} \\ \frac{\partial^2 V}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 V}{\partial \theta_2^2} & \cdots & \frac{\partial^2 V}{\partial \theta_2 \partial \theta_{n_p}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 V}{\partial \theta_{n_p} \partial \theta_1} & \frac{\partial^2 V}{\partial \theta_{n_p} \partial \theta_2} & \cdots & \frac{\partial^2 V}{\partial \theta_{n_p}^2} \end{bmatrix}$$

the Hessian of the function $V(\theta)$. Starting from an initial guess $\theta(0)$, an iterative search for the best estimate of the parameter vector at iteration step l generally takes the form

$$\theta(l+1) = \theta(l) + \alpha f(l) = \theta(l) + \Delta\theta(l) \tag{4.2}$$

where $f(l) \in \mathbb{R}^{n_p}$ is called the *search direction* at iteration step l and the constant $\alpha > 0$ is called the *learning rate*. One can distinguish between methods that use only V and ∇V for updating the estimate (e.g. the *steepest descent method*), and methods that use V, ∇V and $\nabla^2 V$ (*Newton methods*).

Before we discuss steepest descent and Newton methods and their application to the training of MLP networks, we briefly recall some useful definitions and facts.

Directional Derivative

The elements $\partial V/\partial \theta_i$ of the gradient vector and $\partial^2 V/\partial \theta_i^2$ of the diagonal of the Hessian are the first and second derivative, respectively, of V along the θ_i axis. The first and second derivatives along the direction of an arbitrary vector f - the *directional derivatives* along f - are given by

$$\frac{f^T \nabla V(\theta)}{\|f\|} \quad \text{and} \quad \frac{f^T \nabla^2 V(\theta) f}{\|f\|}$$

respectively.

Example 4.1

Consider the function

$$V(\theta) = \theta_1^2 + 9\theta_2^2, \qquad \theta^0 = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix}, \qquad f = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

The gradient of $V(\theta)$ at θ^0 is

$$\nabla V|_{\theta^0} = \left.\begin{bmatrix} 2\theta_1 \\ 18\theta_2 \end{bmatrix}\right|_{\theta^0} = \begin{bmatrix} 3 \\ 9 \end{bmatrix}$$

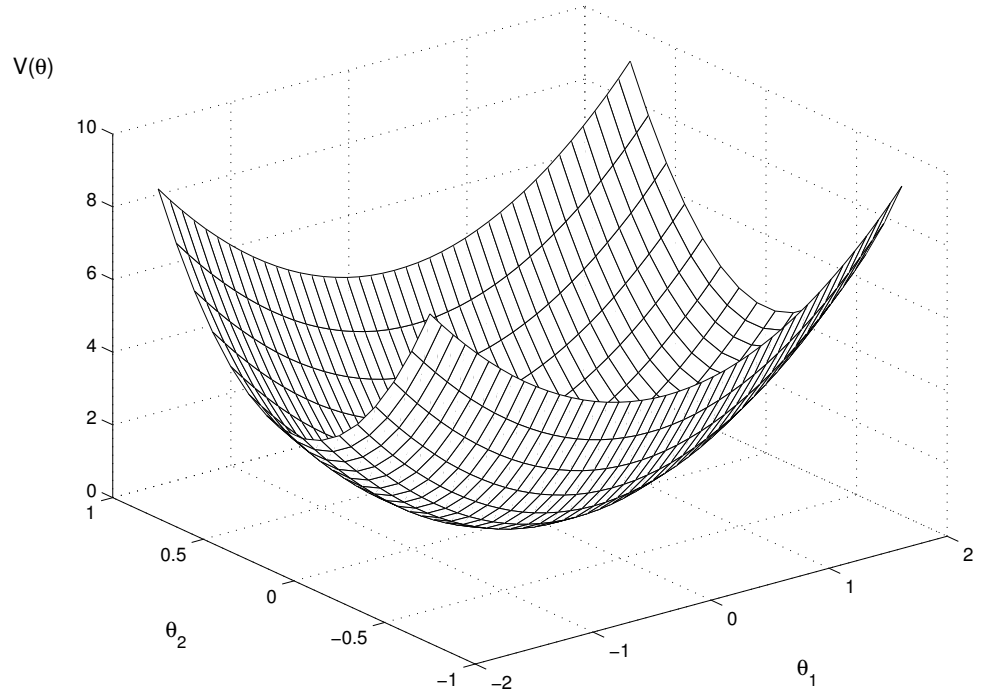

Figure 4.1: Surface plot of $V(\theta)$

The directional derivative along f is therefore

$$\frac{1}{\|f\|}[3 \quad -1]\begin{bmatrix}3\\9\end{bmatrix} = 0$$

The function V is shown in Figure 4.1 as surface plotted over the $\theta_1 - \theta_2$ plane, and in Figure 4.2 as a contour plot. The directional derivative along f is zero because f points in the direction of the tangent of the level curve at θ^0 - in this direction the slope is zero. The slope has its maximum value in the direction of the gradient, and its minimum value in the opposite direction.

Conditions for Minima

In the present context, the objective of performance learning is to minimize the network performance index $V(\theta)$. A point θ^0 is called

- a *strong minimum* of $V(\theta)$ if a scalar β exists such that $V(\theta^0) < V(\theta^0 + \Delta\theta)$ for all $\Delta\theta$ such that $\beta > \|\Delta\theta\| > 0$
- a *weak minimum* of $V(\theta)$ if it is not a strong minimum and if a scalar β exists such that $V(\theta^0) \leq V(\theta^0 + \Delta\theta)$ for all $\Delta\theta$ such that $\beta > \|\Delta\theta\| > 0$
- a *global minimum* of $V(\theta)$ if $V(\theta^0) < V(\theta^0 + \Delta\theta)$ for all $\Delta\theta \neq 0$.

A linear approximation of $V(\theta)$ in the neighborhood of θ^0 is

$$V(\theta) = V(\theta^0 + \Delta\theta) \approx V(\theta^0) + \nabla V(\theta)^T|_{\theta^0}\Delta\theta$$

A necessary (but not sufficient) condition for θ^0 to be a strong minimum is

$$\nabla V(\theta)|_{\theta^0} = 0$$

Points satisfying this condition are called *stationary points* of $V(\theta)$. Whether or not a stationary point $V(\theta)$ is a minimum depends on the higher order terms of the Taylor series expansion

$$V(\theta^0 + \Delta\theta) = V(\theta^0) + \frac{1}{2}\Delta\theta^T\nabla^2 V(\theta)\Big|_{\theta^0}\Delta\theta + \dots$$

In a small neighborhood of θ^0 we may neglect third and higher order terms, and a sufficient condition for θ^0 to be a strong minimum is

$$\nabla V^2(\theta)|_{\theta^0} > 0$$

i.e. the Hessian at θ^0 is positive definite. Note however that this is not a necessary condition, since θ^0 can still be a strong minimum even if the second order term in the Taylor series is zero (e.g. when the third order term is positive).

Quadratic Functions

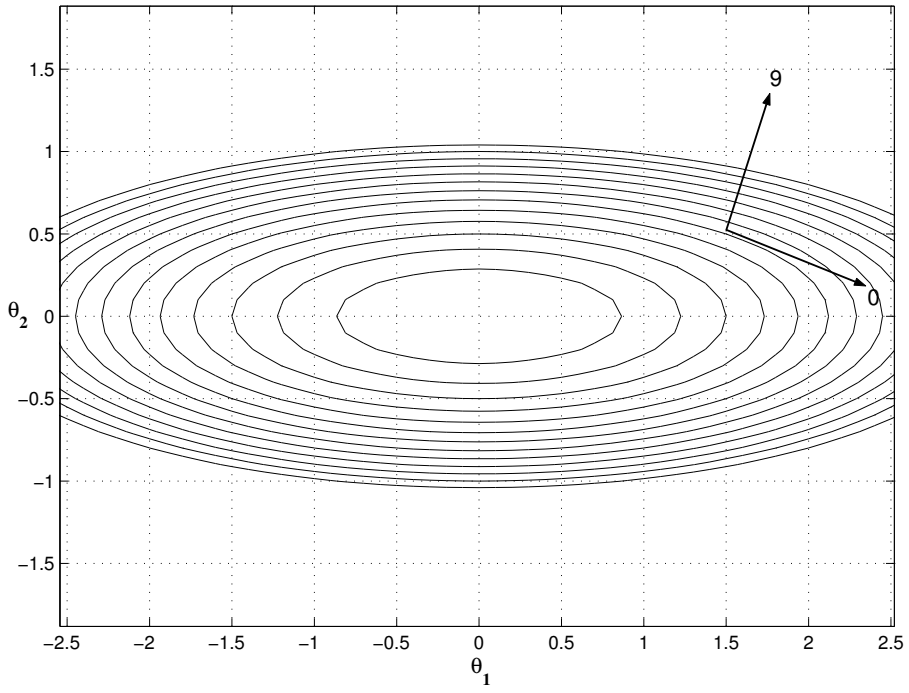

Figure 4.2: Contour plot of $V(\theta)$

Quadratic functions are of particular interest when studying the optimization of quadratic performance indices. A quadratic function has the form

$$F(x) = \frac{1}{2}x^T Q x + p^T x + r \tag{4.3}$$

where F is scalar function of $x \in \mathbb{R}^n$, $Q = Q^T \in \mathbb{R}^{n \times n}$ and p and r are a column vector and a scalar, respectively. We have

$$\nabla F(x) = Qx + p \quad \text{and} \quad \nabla^2 F(x) = Q$$

In Exercise 5.1 it is shown that the eigenvalues of the Hessian Q are the second derivatives of $F(x)$ in the direction of the corresponding eigenvectors. The following three quadratic functions all have a stationary point at $x = 0$; they illustrate how the Hessian determines the character of the stationary point.

Example 4.2

The Hessian of the quadratic function

$$F(x) = \frac{1}{2}x^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} x$$

has eigenvalues 3 and 1 and is positive definite. Thus, the origin is a strong minimum. This function is shown in Figure 4.3. The eigenvectors of the Hessian point in the direction of the principal axes of the ellipse-shaped level curves; the second derivatives (curvature) in theses directions are 3 and 1.

Example 4.3

The Hessian of

$$F(x) = \frac{1}{2}x^T \begin{bmatrix} -1 & -6 \\ -6 & -1 \end{bmatrix} x$$

has eigenvalues 5 and -7 and is indefinite. The function is shown in Figure 4.4. The stationary point $\theta = 0$ is a minimum in the direction of the eigenvector corresponding to eigenvalue 5, and a maximum in the direction of the eigenvector corresponding to eigenvalue -7. Such a point is called a *saddle point*.

Example 4.4

The Hessian of

$$F(x) = \frac{1}{2}x^T \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} x$$

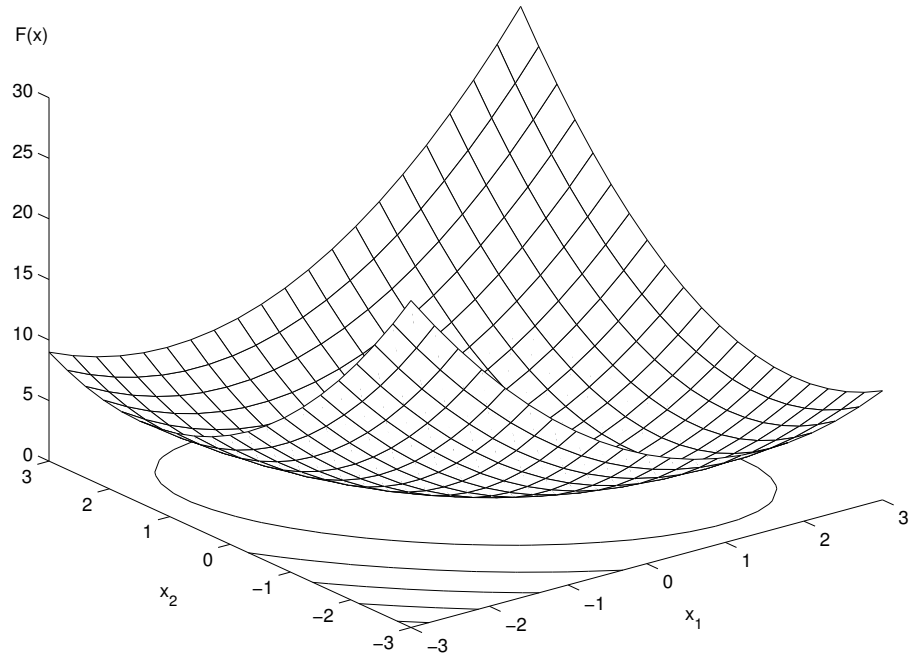

Figure 4.3: Strong minimum

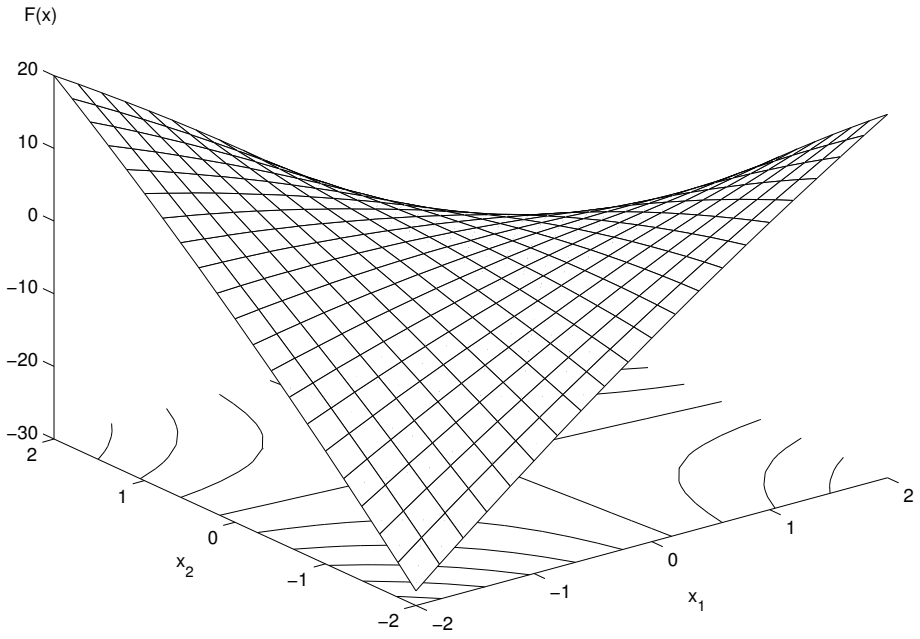

Figure 4.4: Saddle point

has eigenvalues 2 and 0 and is positive semidefinite. The function is shown in Figure 4.5, the origin is a weak minimum. The valley has the direction of the eigenvector corresponding to the zero eigenvalue.

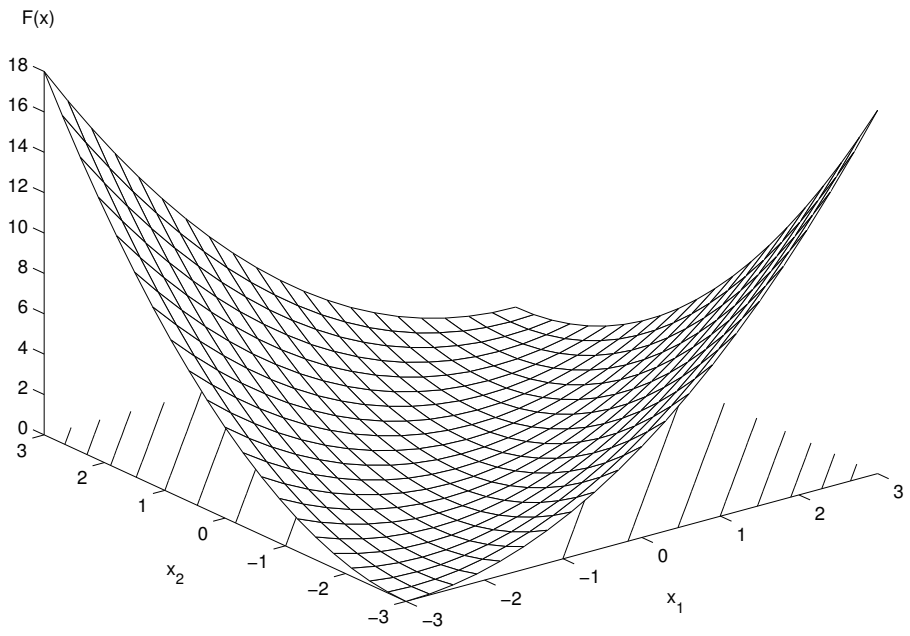

Figure 4.5: Weak minimum

Steepest Descent Method

To simplify notation, we let

$$g(l) = \nabla V(\theta)|_{\theta(l)}$$

denote the gradient at iteration step l, and

$$H(l) = \nabla^2 V(\theta)|_{\theta(l)}$$

denote the Hessian at iteration step l. A first order approximation of the value of the cost function at iteration step $l+1$ is then

$$V(\theta(l+1)) = V(\theta(l) + \Delta\theta(l)) \approx V(\theta(l)) + g^T(l)\Delta\theta(l) = V(\theta(l)) + \alpha g^T(l) f(l)$$

Since $\alpha > 0$ we need $g^T(l)f(l) < 0$, and a search direction satisfying this condition is called a *descent direction*. The steepest descent is obtained in the direction of the negative gradient, i.e. $f(l) = -g(l)$. With this search direction, the iterative search in (4.2) turns into the *steepest descent method*

$$\theta(l+1) = \theta(l) - \alpha g(l) \tag{4.4}$$

Example 4.1 (*continued*)

With

$$V = \theta_1^2 + 9\theta_2^2 \quad \text{and} \quad \theta(0) = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix}$$

we have $g(0) = [3 \;\; 9]^T$. With a learning rate $\alpha = 0.02$, the first iteration step thus yields $\theta(1) = [1.44 \;\; 0.32]$. Figure 4.6 shows the trajectory resulting from 50 iterations of (4.4); note that the steps become increasingly smaller as the minimum is approached. This is a consequence of the learning rate being small. The trajectories obtained with learning rates $\alpha = 0.110$ and $\alpha = 0.112$, are shown in Figures 4.7 and 4.8, respectively. With the last choice of α, the iteration does not converge.

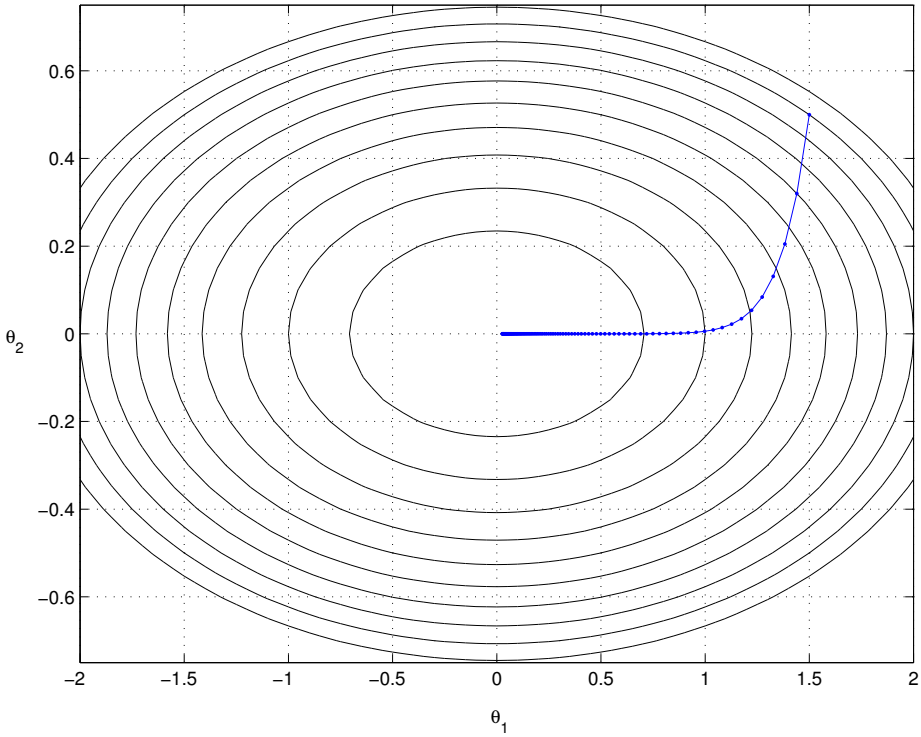

Figure 4.6: Trajectory with $\alpha = 0.02$

The above example illustrates the following relationship - derived in Exercise 5.2 - between the maximum eigenvalue of the Hessian of a quadratic function, and an upper bound on the learning rate: If the cost function is quadratic, the steepest descent algorithm converges if

$$\alpha < \frac{2}{\lambda_{\max}(H)}$$

where $\lambda_{\max}(H)$ denotes the maximum eigenvalue of the Hessian. In the example we have

$$\nabla^2 V = \begin{bmatrix} 2 & 0 \\ 0 & 18 \end{bmatrix} \quad \Rightarrow \quad \alpha < \frac{2}{18} = 0.111$$

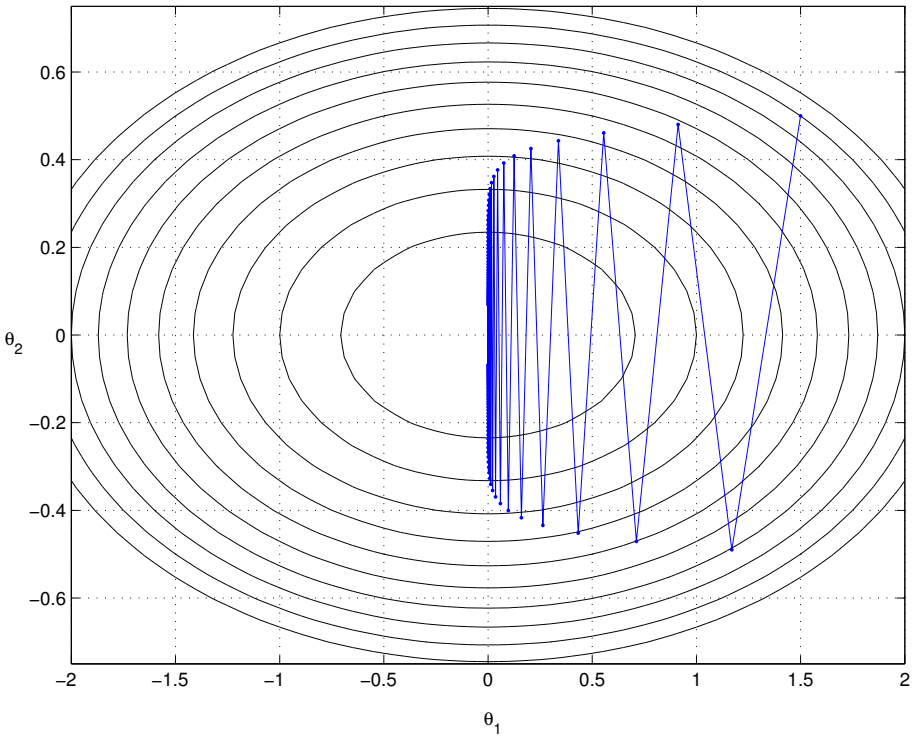

Figure 4.7: Trajectory with $\alpha = 0.110$

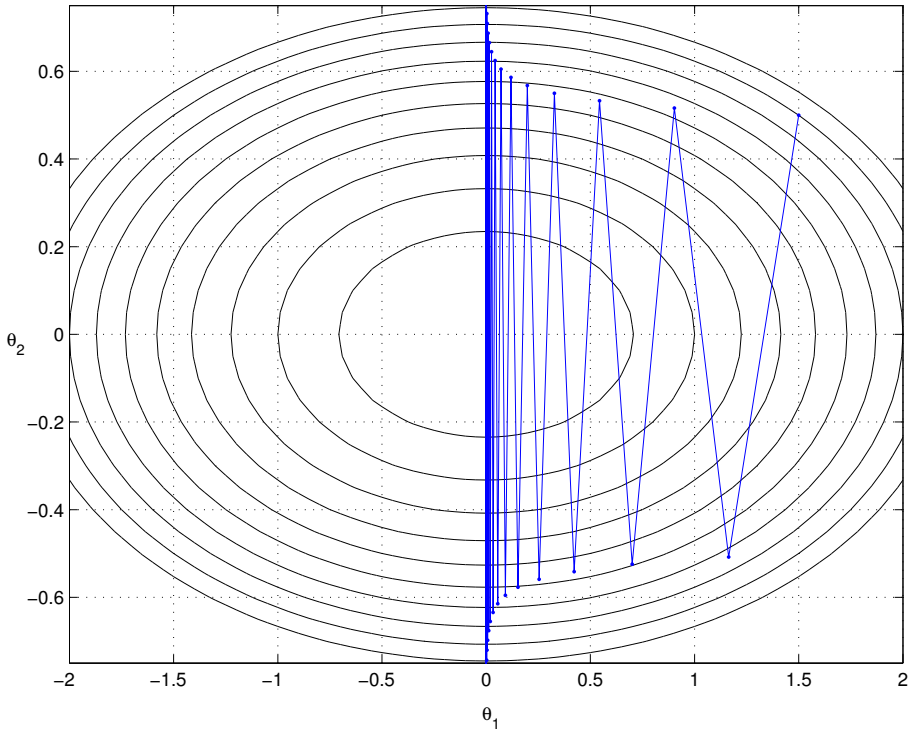

Figure 4.8: Trajectory with $\alpha = 0.112$

The last example also illustrates that convergence is fast in the direction of eigenvectors corresponding to large eigenvalues, and small in the direction of eigenvectors corresponding to small eigenvalues. Obviously the convergence properties of the steepest descent algorithm as in (4.4) are unsatisfactory. Improvements are possible by allowing a variable learning rate, e.g. by searching along the gradient direction for the minimum of the cost function (*line search*), or by modifying the search directions. Here we will however not discuss improvements of the basic steepest descent method, but we will now turn to a method that considers not only the gradient but also the Hessian of the cost function.

Newton's Method

A second order approximation of the value of the cost function at iteration step $l+1$ is

$$V(\theta(l+1)) = V(\theta(l) + \Delta\theta(l)) \approx V(\theta(l)) + g^T(l)\Delta\theta(l) + \frac{1}{2}\Delta\theta^T(l)H(l)\Delta\theta(l)$$

Let $F(\Delta\theta(l))$ denote the right hand side of the above approximation, seen as a quadratic function of $\Delta\theta(l)$. To find the stationary point of this function we solve

$$\frac{\partial V(\Delta\theta(l))}{\partial\Delta\theta(l)} = g(l) + H(l)\Delta\theta(l) = 0$$

for

$$\Delta\theta(l) = -H^{-1}(l)g(l)$$

This leads to *Newton's method* for updating the estimate

$$\theta(l+1) = \theta(l) - H^{-1}(l)g(l) \tag{4.5}$$

Example 4.1 *(continued)*

With

$$g(0) = \begin{bmatrix} 3 \\ 9 \end{bmatrix} \quad \text{and} \quad H(0) = \begin{bmatrix} 2 & 0 \\ 0 & 18 \end{bmatrix}$$

we obtain in the first iteration step

$$\theta(1) = \begin{bmatrix} 1.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{18} \end{bmatrix} \begin{bmatrix} 3 \\ 9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Thus, Newton's method converges in one step. This is of course due to the fact that the cost function in this example is quadratic. The cost functions we encounter when training neural networks are in general not quadratic in the network parameters. As an illustration, Figure 4.9 shows the cost function for the multilayer perceptron network in Exercise 3.1, when only two of the seven network parameters (w_{11}^1 and w_{11}^2) are varied, while the other parameters are fixed at their optimal values. A typical feature are the

long valleys where the gradient is small, and where the steepest descent method would lead to very slow progress.

In comparison with the steepest descent algorithm, Newton's method is generally faster when the search is close to a minimum. On the other hand, far from a minimum Newton's method can lead to unpredictable results. Moreover, it is computationally much more expensive, because it requires the computation of $H(l)$ and of $H^{-1}(l)$ at each step. As a consequence, one would prefer

- Steepest descent when far from a minimum
- Newton's method when close to a minimum.

We will see later that this idea is realized in the *Levenberg-Marquardt algorithm.*

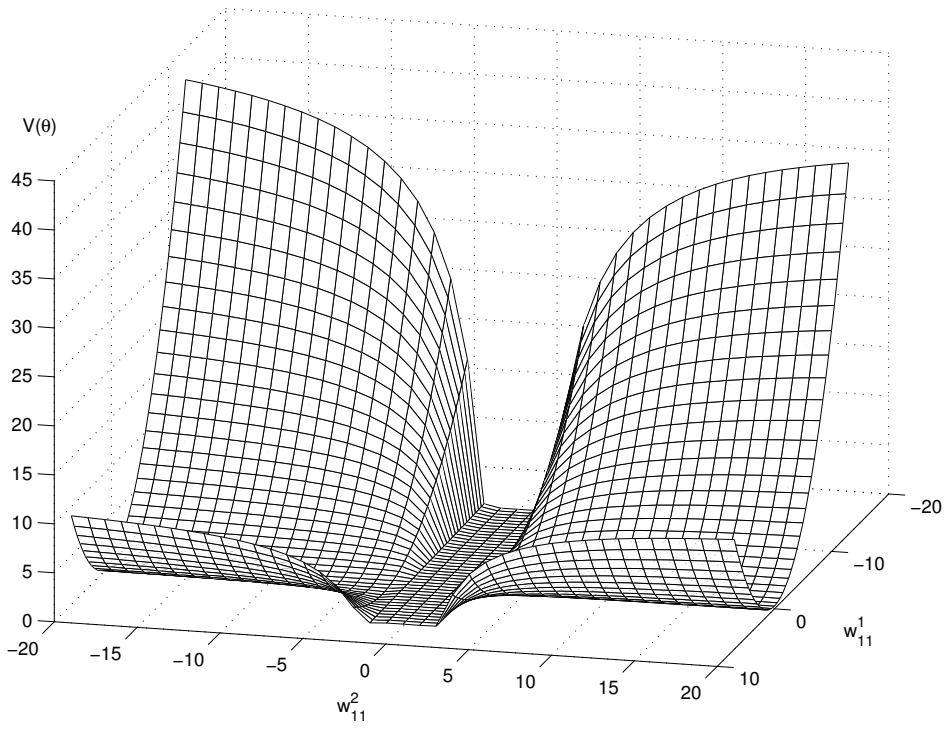

Figure 4.9: Cost function for MLP from Exercise 3.1 with w^1_{11} and w^1_{12} as variables

Chapter 5

Training Neural Networks Levenberg-Marquardt Backpropagation

We will now apply the steepest descent method to train the multilayer perceptron network shown in Figure 5.1. The network output is denoted $\hat{y}$ to indicate that the network will be used for nonlinear system identification; we have

$$\hat{y}(k|\theta) = f^2(W^2 f^1(W^1\varphi(k) + w_0^1) + w_0^2)$$

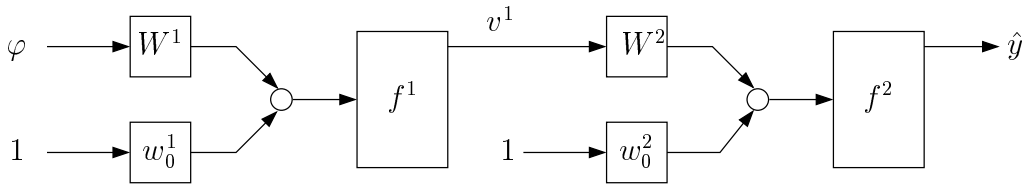

Figure 5.1: Multilayer perceptron network

The i^{th} element of the gradient $\nabla V(\theta)$ of the performance index (4.1) is

$$\frac{\partial V(\theta)}{\partial \theta_i} = \frac{1}{2N}\sum_{k=1}^{N}\frac{\partial(y(k) - \hat{y}(k|\theta))^2}{\partial \theta_i} = -\frac{1}{N}\sum_{k=1}^{N}\frac{\partial \hat{y}(k|\theta)}{\partial \theta_i}\varepsilon(k|\theta) \tag{5.1}$$

where $\varepsilon(k|\theta)$ is the prediction error at time k. The gradient of the cost function V is therefore the negative average of the gradients of the predicted output at all sampling instants, weighted by the prediction error at each sampling instant.

Output Layer

We will now compute the derivative of the predicted output with respect to each weight and bias parameter, beginning with the output layer. Here we have

$$\hat{y}(k|\theta) = f^2\left(\sum_{j=0}^{s^1} w_{1j}^2 v_j^1(k)\right) = f^2(h^2(k))$$

Note that the summation begins at $j = 0$: by defining $v_0^1(k) = 1$ we can treat the bias term w_{10}^2 as an additional weight parameter. Applying the chain rule, we obtain

$$\frac{\partial \hat{y}(k|\theta)}{\partial w_{1j}^2} = \left.\frac{\partial f^2(h^2)}{\partial h^2}\right|_k v_j^1(k)$$

If we define the *sensitivity*

$$\delta_1^2(k) = \left.\frac{\partial f^2(h^2)}{\partial h^2}\right|_k \tag{5.2}$$

of the first (and in this case only) output of layer 2 at sampling instant k, we can write

$$\frac{\partial \hat{y}(k|\theta)}{\partial w_{1j}^2} = \delta_1^2(k) v_j^1(k) \tag{5.3}$$

Hidden Layer

The derivative of the predicted output with respect to the weight and bias parameters in the hidden layer can be obtained from

$$\hat{y}(k|\theta) = f^2\left(\sum_{j=1}^{s^1} w_{1j}^2 f^1\left(\sum_{i=0}^{r} w_{ji}^1 \varphi_i(k)\right) + w_0^2\right)$$

The index i points to input channels, while the index j points to neurons in the hidden layer. Note that the inner summation begins at $i = 0$; we define $\varphi_0(k) = 1$ so that the bias terms of the hidden layer can be treated as additional weights.

Applying the chain rule yields

$$\frac{\partial \hat{y}(k|\theta)}{\partial w_{ji}^1} = \left.\frac{\partial f^2(h^2)}{\partial h^2}\right|_k \left.\frac{\partial h^2}{\partial h^1}\right|_k \left.\frac{\partial h^1}{\partial w_{ji}^1}\right|_k$$

Observing that

$$\left.\frac{\partial h^1}{\partial w_{ji}^1}\right|_k = \varphi_i(k)$$

and

$$\left.\frac{\partial h^2}{\partial h^1}\right|_k = w_{1j}^2 \left.\frac{\partial f^1(h^1)}{\partial h^1}\right|_k$$

this can be written as

$$\frac{\partial \hat{y}(k|\theta)}{\partial w_{ji}^1} = \delta_j^1(k)\varphi_i(k) \tag{5.4}$$

where we defined the sensitivity of the j^{th} output in the first layer at time k

$$\delta_j^1(k) = \delta_1^2(k)\, w_{1j}^2 \frac{\partial f^1(h^1)}{\partial h^1}\bigg|_k \tag{5.5}$$

Comparing (5.3) with (5.4), we see that the derivatives in each layer are given by the corresponding sensitivity multiplied with the input of the respective layer. Starting at the output layer, we can compute its sensitivity from (5.2) and obtain the corresponding derivatives from (5.3). Having computed the output sensitivity, we can use (5.5) to calculate the sensitivity of the hidden layer; the derivatives with respect to weights in that layer are then obtained from (5.4). If we have a network with more than two layers, we can proceed in the same manner: the sensitivity of layer m can be computed from the sensitivity of layer $m+1$; the derivatives of the predicted output with respect to weights in each layer are thus obtained by *backpropagating* the sensitivities.

Training a Two-Layer Sig-Lin Perceptron Network

We have shown in this section how the gradient of the sum of squared prediction errors (2.1) with respect to the weights and biases of a multilayer perceptron network can be computed by backpropagating the sensitivities. To illustrate this approach, we will apply it to a two-layer sig-lin network, i.e. a network as in Figure 5.1 where f^1 is a sigmoid and f^2 a linear activation function. Assume that at the l^{th} iteration our estimate of the parameter vector is $\theta(l)$. Using the steepest descent method, the update of the estimate of the i^{th} parameter is then

$$\theta_i(l+1) = \theta_i(l) - \alpha \left.\frac{\partial V}{\partial \theta_i}\right|_l \tag{5.6}$$

Substituting (5.1) for the partial derivative of V, we see that we need the weighted average of the partial derivatives of the predicted output over all sampling instants. The sensitivity of the linear output layer at sampling instant k is

$$\delta_1^2(k) = \left.\frac{\partial f^2(h^2)}{\partial h^2}\right|_k = 1 \tag{5.7}$$

and thus

$$\frac{\partial \hat{y}(k|\theta)}{\partial w_{1j}^2} = \delta_1^2(k) v_j^1(k) = v_j^1(k)$$

If we have log-sig activation functions

$$f^1(h) = \frac{1}{1+e^{-h}}$$

in the hidden layer, where for simplicity we let h denote the weighted sum of inputs h_j^1 at the j^{th} hidden neuron, we obtain

$$\frac{\partial f_j^1(h)}{\partial h} = \frac{\partial}{\partial h}\frac{1}{1+e^{-h}} = \frac{e^{-h}}{(1+e^{-h})^2} = \left(1 - \frac{1}{1+e^{-h}}\right)\frac{1}{1+e^{-h}}$$

The sensitivity at the j^{th} hidden neuron is therefore

$$\delta_j^1(k) = w_{1j}^2 \left(1 - v_j^1(k)\right) v_j^1(k) \tag{5.8}$$

where w_{1j}^2 refers to elements of $W^2(l)$ obtained at the previous iteration step.

Updating the Weights

The weight matrices W^m (where $m = 1$ represents the hidden layer and $m = 2$ the output layer) are updated at iteration step l according to

$$W^m(l+1)m = W^m(l) - \alpha \Delta W^m(l)$$

For the output layer we have, using (5.7)

$$\Delta W^2(l) = -\frac{1}{N} \sum_{k=1}^{N} {v^1}^T(k)\varepsilon(k)$$

Note that because $v_0^1 = 1$ the bias update is equal to the negative average of prediction errors

$$\Delta w_0^2(l) = -\frac{1}{N} \sum_{k=1}^{N} \varepsilon(k)$$

Using (5.8), we obtain for the hidden layer update

$$\Delta W^1(l) = -\frac{1}{N} \sum_{k=1}^{N} \delta^1(k)\varphi^T(k)\varepsilon(k)$$

and the bias update

$$\Delta w_0^1(l) = -\frac{1}{N} \sum_{k=1}^{N} \delta^1(k)\varepsilon(k)$$

Levenberg-Marquardt Backpropagation

In the previous section it was shown how a multilayer perceptron network can in principle be trained by steepest descent backpropagation. In practice it turns out, unfortunately, that steepest descent backpropagation is far too slow to be of practical value. Recall that close to minimum Newton methods are much more efficient than steepest descent. The most widely used method for training MLP networks is the Levenberg-Marquardt backpropagating technique, a clever combination of steepest descent and an approximate Newton method.

A Newton update of the parameter estimate is given by

$$\theta(l+1) = \theta(l) - H^{-1}(l)g(l) \tag{5.9}$$

where as before $H(l)$ denotes the Hessian and $g(l)$ the gradient at iteration step l of the cost

$$V(\theta) = \frac{1}{2N}\sum \varepsilon^2(k,\theta) = \frac{1}{2N}E^T(\theta)E(\theta)$$

where

$$E(\theta) = [\varepsilon(1)\ \ \varepsilon(2)\ \ \ldots\ \ \varepsilon(N)]^T$$

is a column vector containing all prediction errors obtained with parameter vector θ. Recalling that the i^{th} element of the gradient vector is

$$\frac{\partial V}{\partial \theta_i} = \frac{1}{N}\sum_{k=1}^{N} \varepsilon(k,\theta)\frac{\partial \varepsilon(k,\theta)}{\partial \theta_i}$$

we observe that the gradient can be written as

$$\nabla V(\theta) = \frac{1}{N}\begin{bmatrix} \frac{\partial \varepsilon_1}{\partial \theta_1} & \cdots & \frac{\partial \varepsilon_N}{\partial \theta_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_1}{\partial \theta_{n_p}} & \cdots & \frac{\partial \varepsilon_N}{\partial \theta_{n_p}} \end{bmatrix}\begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_N \end{bmatrix} = \frac{1}{N}J^T(\theta)E(\theta) \tag{5.10}$$

where we introduced the Jacobian $J(\theta)$ of the prediction error vector $E(\theta)$. To obtain a similar representation of the Hessian, consider the (i,j) element

$$\left(\nabla^2 V\right)_{ij} = \frac{\partial^2 V}{\partial \theta_i \partial \theta_j} = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{2}\frac{\partial^2 \varepsilon^2(k)}{\partial \theta_i \partial \theta_j}$$

For the term under the sum we have (suppressing dependence on θ in notation)

$$\frac{1}{2}\frac{\partial^2 \varepsilon^2(k)}{\partial \theta_i \partial \theta_j} = \frac{\partial}{\partial \theta_j}\left(\frac{1}{2}\frac{\partial}{\partial \theta_i}\varepsilon^2(k)\right) = \frac{\partial}{\partial \theta_j}\left(\varepsilon(k)\frac{\partial \varepsilon(k)}{\partial \theta_i}\right) = \frac{\partial \varepsilon(k)}{\partial \theta_j}\frac{\partial \varepsilon(k)}{\partial \theta_i} + \varepsilon(k)\frac{\partial^2 \varepsilon(k)}{\partial \theta_i \partial \theta_j}$$

The Hessian of $V(\theta)$ can therefore be written as

$$\nabla^2 V(\theta) = \frac{1}{N}\left(J^T(\theta)J(\theta) + S(\theta)\right) \tag{5.11}$$

where

$$S(\theta) = \sum_{k=1}^{N}\varepsilon(k)\nabla^2 \varepsilon(k)$$

Gauss-Newton Method

Based on (5.11), a frequently used approximation of the Hessian is

$$\nabla^2 V \approx \frac{1}{N}J^T(\theta)J(\theta)$$

This approximation is known as the *Gauss-Newton Hessian*. It has the advantage that it does not require the computation of second derivatives. To justify the use of the Gauss-Newton Hessian, consider a linear approximation of the prediction error around a given value θ^0 of the parameter vector

$$\tilde{\varepsilon}(k,\theta) = \varepsilon(k,\theta^0) + \nabla^T \varepsilon(k,\theta)(\theta - \theta^0)$$

leading to an approximate cost function

$$\tilde{V}(\theta) = \frac{1}{2N}\sum_{k=1}^{N} \tilde{\varepsilon}^2(k,\theta)$$

It is straightforward to check that

$$\nabla \tilde{V}(\theta)\Big|_{\theta=\theta_0} = \frac{1}{N} J^T(\theta^0) E(\theta^0)$$

and

$$\nabla^2 \tilde{V}(\theta)\Big|_{\theta=\theta_0} = \frac{1}{N} J^T(\theta^0) J(\theta^0)$$

Thus, while the gradient of the approximate cost is equal to the true gradient at θ^0 (not surprisingly since we use a linear approximation), the Hessian of the approximate cost turns out to be just the Gauss-Newton Hessian.

Replacing the Hessian in (5.9) by the Gauss-Newton Hessian and using (5.10) and (5.11) yields

$$\theta(l+1) = \theta(l) - (J_l^T J_l)^{-1} J_l^T E_l$$

where we use the shorthand notation $J_l = J(\theta(l))$ and $E_l = E(\theta(l))$. The resulting search direction is called a *Gauss-Newton direction*.

Levenberg-Marquardt Algorithm

Note that at each iteration step we have $J_l^T J_l \geq 0$. If the Gauss-Newton Hessian is near singular, one can improve its numerical condition by replacing it with $J_l^T J_l + \mu I$ to get

$$\theta(l+1) = \theta(l) - (J_l^T J_l + \mu_l I)^{-1} J_l^T E_l \tag{5.12}$$

where μ is a small positive constant. Moreover, one can replace μ by a time-varying parameter μ_l that can be used for tuning the search direction to be either close to the steepest descent direction or close to the Gauss-Newton direction at a given iteration step: if μ_l is large we have

$$\theta(l+1) \approx \theta(l) - \frac{1}{\mu_l} J_l^T E_l = \theta(l) - \frac{N}{\mu_l} g(l)$$

In this case the search direction is close to the steepest descent direction. On the other hand, if μ_l is small the search direction in (5.12) turns into the Gauss-Newton direction. This observation is the basis of the Levenberg-Marquardt algorithm: choose a positive constant $\rho > 1$ (e.g. $\rho = 10$) and a small positive value μ_0 (e.g. $\mu_0 = 0.01$), and repeat the following:

- At iteration step l update θ according to (5.12), compute $V(\theta_{l+1})$
- If $V(\theta_{l+1}) \geq V(\theta_l)$, reject update and replace μ_l by $\rho\mu_l$, repeat iteration step l
- If $V(\theta_{l+1}) < V(\theta_l)$, accept update, set $\mu_{l+1} = \mu_l/\rho$, go to iteration step $l+1$.

This algorithm tries to use a Gauss-Newton direction when possible but returns to steepest descent when the Gauss-Newton direction would give no improvement. The search directions are illustrated in Figure 5.2.

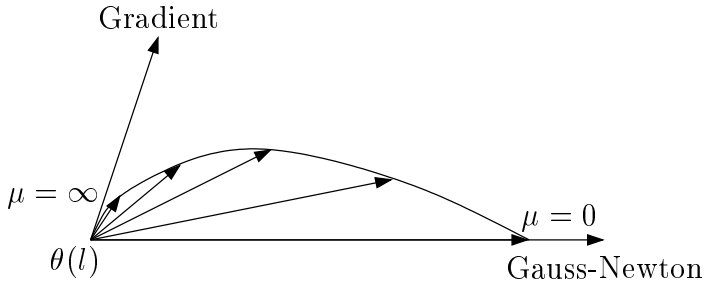

Figure 5.2: Search direction in Levenberg-Marquardt algorithm

Backpropagation

The efficiency of the steepest descent backpropagation discussed in the previous section can be improved when the steepest descent direction in (5.6) is replaced by a search direction determined by the Levenberg-Marquardt algorithm. Computationally all that is needed is the Jacobian $J(\theta)$, which in turn requires to compute

$$\frac{\partial \varepsilon(k,\theta)}{\partial \theta_i} = -\frac{\partial \hat{y}(k,\theta)}{\partial \theta_i}$$

for $k = 1, \ldots, N$ and $i = 1, \ldots, n_p$. These quantities can be computed from (5.2) and (5.5) using backpropagation.

Nonlinear System Identification Using MLP

Having discussed how multilayer perceptron networks can be trained to approximate a nonlinear function $y = g(\varphi)$ based on a set of data pairs $Z^N = \{y(k), \varphi(k); k = 1, \ldots, N\}$, where the regressor vector $\varphi(k)$ contains past input and output data, we now return to the problem of identifying a model for a nonlinear dynamic system. Recall that we are looking for a nonlinear regressor model of the form (2.18)

$$y(k) = g(\varphi(k), \theta) + e(k)$$

Here $y(k)$ represents measured system outputs, the regressor vector $\varphi(k) \in \mathbb{R}^r$ contains samples of measured input and output signals taken prior to sampling instant k, and $\theta \in \mathbb{R}^{n_p}$ is a vector whose elements are the weights and biases of the MLP network to be trained. The nonlinear function $g(\cdot)$ describes the mapping of neural network inputs into outputs; this mapping depends on the weights in θ. Recall also that in the above regressor model we have already transformed the perturbation $v(k)$ in Figure 2.1 into the form

$$v(k) = e(k) + \text{past values of } v$$

where $e(k)$ is a white noise process. In other words, the disturbance acting on the measured plant output may be non-white, but a noise model taking account of this is absorbed into the function $g(\cdot)$ and is represented together with the plant dynamics by the parameter vector θ. As a consequence, the prediction error is white when θ takes its optimal value, compare (2.12). This leads to the predictor model

$$\hat{y}(k) = g(\varphi(k), \theta) \tag{5.13}$$

NNARX Model Structure

The simplest neural network based model structure that can be used to represent the predictor model (5.13) is the NNARX structure, where

$$\varphi(k) = [y_{k-1} \ \ldots \ y_{k-n} \ \ u_{k-d} \ \ldots \ u_{k-d-m}]^T$$

This structure was introduced in Section 2 and is shown again in Figure 5.3. Note that a *static* feedforward network is used to represent a *dynamic* nonlinear system as predictor model. In *training mode*, such a predictor model is used to find the weights and biases that give the best fit between predicted and measured output: at iteration step l, regression vectors $\varphi(k)$ are constructed for $k = p, \ldots, N$ from measured data, applied to the network, and the network outputs $\hat{y}(k)$ obtained with weights and biases according to $\theta(l)$ are compared with the measured outputs $y(k)$. The resulting values of $\hat{y}(k)$ and $\varepsilon(k)$ are used in Levenberg-Marquardt backpropagation to compute a search direction and an update $\Delta\theta(l)$, which provides the weights for iteration step $l + 1$.

In *recall mode*, when a trained network is used to simulate system behaviour, the past values of $y(k)$ which are used as network inputs in $\varepsilon(k)$, are previous network outputs. This implies feedback from network output to input and enables the network to represent dynamic behaviour.

NNARMAX Model Structure

As discussed earlier, the NNARMAX model structure allows a more flexible disturbance model, which can lead to a more accurate model of the system to be identified when

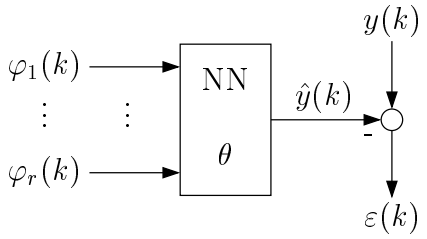

Figure 5.3: NNARX model structure

measurements are corrupted by colored noise. The regressor vector for an NNARMAX model is

$$\varphi(k,\theta) = [y_{k-1} \ \dots \ y_{k-n} \ \ u_{k-d} \ \dots \ u_{k-d-m} \ \ \varepsilon_{k-1} \ \dots \ \varepsilon_{k-n}]^T$$

Note that here even in training mode the predictor model includes feedback from network output to inputs, as shown in Figure 5.4.

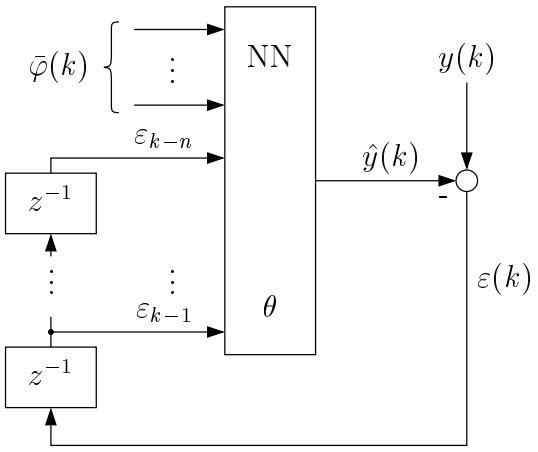

Figure 5.4: NNARMAX model structure

As a consequence, the regressor vector depends - in contrast to NNARX models - on the network parameters θ, thus instead of $\hat{y}(k) = g(\varphi(k), \theta)$ we have

$$\hat{y}(k) = g(\varphi(k,\theta), \theta)$$

Since the predicted output is in this case not a function of independent variables, the gradient of the cost is determined by the *total derivative*

$$\frac{d\hat{y}(k)}{d\theta} = \frac{\partial\hat{y}(k)}{\partial\theta} + \frac{\partial\hat{y}(k)}{\partial\varepsilon(k-1)}\frac{d\varepsilon(k-1)}{d\theta} + \ldots + \frac{\partial\hat{y}(k)}{\partial\varepsilon(k-n)}\frac{d\varepsilon(k-n)}{d\theta} \tag{5.14}$$

Introducing the notation

$$\psi_k = \frac{d\hat{y}(k)}{d\theta} \quad \text{and} \quad \phi_k = \frac{\partial\hat{y}(k)}{\partial\theta}$$

(5.14) can be written as

$$\psi_k = \phi_k - \frac{\partial \hat{y}(k)}{\partial \varepsilon(k-1)}\,\psi_{k-1} - \ldots - \frac{\partial \hat{y}(k)}{\partial \varepsilon(k-n)}\,\psi_{k-n}$$

The backpropagation technique discussed in the previous section can be used to compute ϕ_k. Computing ψ_k however requires also the partial derivatives of predicted outputs with respect to past prediction errors. The problem simplifies if the NNARMAX structure is modified such that the disturbance model is linear. Such a regressor model is

$$\hat{y}(k) = g\left(\bar{\varphi}(k), \bar{\theta}\right) + \Big(C(z^{-1}) - 1\Big)\varepsilon(k) \tag{5.15}$$

where $\bar{\varphi}$ and $\bar{\theta}$ represent the regressor vector and the network parameters, respectively, when prediction errors are removed as network inputs. The polynomial

$$C(z^{-1}) = 1 + c_1 z^{-1} + \ldots + c_n z^{-n}$$

describes the noise characteristics as in linear ARMAX models. The modified NNARMAX model is shown in Figure 5.5.

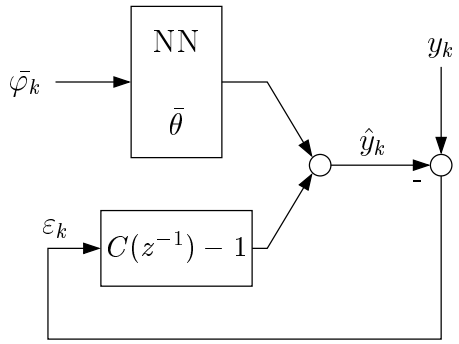

Figure 5.5: Modified NNARMAX model structure

With this modification, (5.14) becomes

$$\psi_k = \phi_k - c_1\psi_{k-1} - \ldots - c_n\psi_{k-n}$$

or

$$\psi_k = \frac{1}{C(z^{-1})}\,\phi_k$$

where

$$\theta = \begin{bmatrix} \bar{\theta} \\ c_1 \\ \vdots \\ c_n \end{bmatrix} \quad \text{and} \quad \phi_k = \begin{bmatrix} \frac{\partial \hat{y}_k}{\partial \bar{\theta}} \\ \varepsilon_{k-1} \\ \vdots \\ \varepsilon_{k-n} \end{bmatrix}$$

Backpropagation can be used to compute $\partial \hat{y}_k / \partial \bar{\theta}$; ψ_k is then computed by forming ϕ_k and filtering it through $1/C(z^{-1})$. The coefficients of $C(z^{-1})$ are contained in θ and

updated in each Levenberg-Marquardt iteration together with the network parameters $\bar{\theta}$. Not that stability of $1/C(z^{-1})$ must be checked at each step; if C is unstable the *spectral factorization theorem* can be used to replace it with a stable polynomial.

Practical Issues

Before a two-layer MLP network is to be trained for a control application, the following choices have to be made

- Sampling time
- Dynamic system order
- Number of hidden neurons
- Training signal

These choices depend on the control objectives and are explored in an exercise in the following section.

Exercises

Problem 5.1

Show that the eigenvalues of the Hessian of a quadratic function are equal to the second derivatives of that function in the direction of the corresponding Hessian eigenvectors.

Problem 5.2

Show that for a quadratic cost function $V(\theta)$, the steepest descent method according to (4.4) will be stable if the learning rate satisfies

$$\alpha < \frac{2}{\lambda_{\max}(\nabla^2 V)}$$

where $\lambda_{\max}(\cdot)$ denotes the maximum eigenvalue of a symmetric matrix.

Chapter 6

Predictive Control Using Neural Networks

In this chapter we discuss the use of neural networks in nonlinear control applications. Neural networks can be used in two ways for controller design, referred to as *direct design* and *indirect design*, respectively. In *direct* neural network based design, the controller itself is a neural network. This means that instead of training a network to learn the behaviour of the plant to be controlled, a network is trained to learn the desired behaviour of the controller. Such techniques have been proposed in the literature, however they have at least two major drawbacks. One is that such control loops are difficult to tune: each re-tuning of the controller requires re-training a neural network. A second disadvantage is due to the fact that direct design usually requires the controller network to learn some form of inverse plant dynamics, which is difficult for plants with unstable or lightly damped inverses.

In *indirect design*, the controller is not a neural network itself, but has access to a neural network as plant model. This approach avoids the difficulties associated with direct design; typical neural network based indirect design techniques are minimum variance control or predictive control. In this section we illustrate the use of neural networks for control with a particular predictive control technique referred to as *approximate predictive control*. However, before we discuss the details of this approach, we will briefly review the idea of predictive control in more general terms.

Predictive Control

It is frequently pointed out that predictive control is the most widely used advanced control scheme in industrial applications ("advanced" here meaning "more sophisticated than PID control"). Applications are mainly in process control, one reason being that predictive control laws may require extensive calculations between two sampling instants,

and in process control sampling is usually sufficiently slow. It can however be expected that the continuing improvement of computational power will lead in the future to the use of predictive control also for “fast applications" like mechatronic systems.

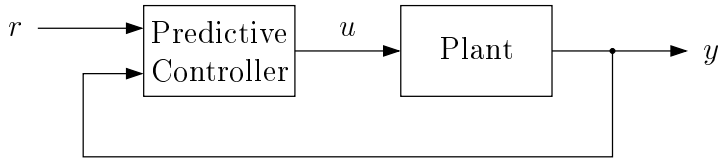

Figure 6.1: Control loop with predictive controller

A control loop with a predictive controller is shown in Figure 6.1. The principal idea of predictive control is to predict plant responses to different control inputs, and use these predictions to optimize the control input at any given sampling instant. Usually a quadratic cost function

$$J_k = \sum_{i=N_1}^{N_2} (r_{k+i} - \hat{y}_{k+i})^2 + \rho \sum_{i=1}^{N_u} \Delta u_{k+i-1}^2 \tag{6.1}$$

is minimized, where control error and control effort are penalized within given time windows - referred to as *prediction horizon* and *control horizon*, respectively.

Typical prediction and control horizons are shown in Figure 6.2. At sampling instant kT the output is predicted for $k + N_1 \leq t/T \leq k + N_2$ as a function of the control input. The changes Δu_{k+i} at $1 \leq i \leq N_u - 1$ of the control input compared to its most recent value u_{k-1} are considered as decision variables in the problem of minimizing (6.1). The prediction horizon N_2 is usually larger than the control horizon N_u, and in this case the assumption

$$\Delta u_{k+i} = 0, \qquad i \geq N_u$$

is made, i.e. the control input is assumed constant beyond the control horizon. Output predictions begin at $k = N_2$, this value is usually taken to be the estimated dead time of the plant.

Receding Horizon

The problem to be solved at time $t = kT$ is

$$\tilde{U}_k^* = \min_{\tilde{U}_k} J_k$$

where we introduced the column vector

$$\tilde{U}_k = [\Delta u_k \;\; \Delta u_{k+1} \;\; \ldots \;\; \Delta u_{k+N_u-1}]^T$$

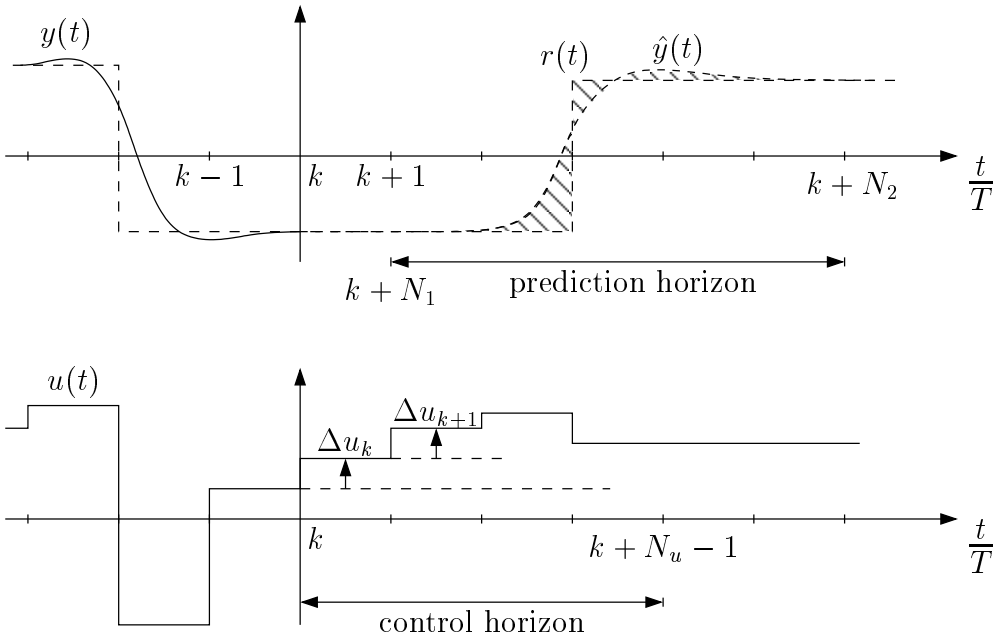

Figure 6.2: Prediction and control horizon

Of the optimal control sequence defined by $\tilde{U}_k^*$, only the first value $u_k = u_{k-1} + \Delta u_k^*$ is applied to the plant. At time $t = kT + T$ the problem

$$\tilde{U}_{k+1}^* = \min_{\tilde{U}_{k+1}} J_{k+1}$$

is solved again, and u_{k+1} is applied. The procedure is repeated at each sampling instant; this technique of using only the first value of a sequence of future control inputs is called a *receding horizon strategy.*

There is a variety of predictive control laws that are used in practice. The plant model used for predicting outputs can be linear or nonlinear; plants can be represented as state space or transfer function models. For nonlinear plants neural network models can be used. One of the strongest features of predictive control is its capability of taking input and output constraints into account when optimizing the control input.

Linear Plant Model - Generalized Predictive Control

Of course the complexity of a predictive control strategy and the required computational effort depend strongly on the choice of model type: nonlinear models require a non-linear optimization problem to be solved on-line in each sampling interval. In particular for “fast applications" only predictive controllers based on linear models are realistic. In this section we introduce a widely used linear predictive control law - known as *generalized predictive control* (GPC) - and in the following section we will see how this control law

can be combined with a nonlinear neural network model by extracting a linearized plant model from the network at each sampling instant.

GPC is used with a slight modification of either the ARX or the ARMAX model structure. For simplicity we consider ARX models

$$A(z^{-1})y(k) = B(z^{-1})u(k-1) + e(k) + \zeta(k) \tag{6.2}$$

where $d = 1$ is assumed for plant dead time. The term $e(k)$ represents zero mean white noise as before, and an extra term $\zeta(k)$ has been added to model step disturbances or (as will become clear in the following section) offset values resulting from linearization. Both white noise and offset together are often represented by a single signal that is generated by integrating white noise, leading to the *ARIX model structure*

$$A(z^{-1})y(k) = B(z^{-1})u(k-1) + \frac{1}{1-z^{-1}}\, e(k) \tag{6.3}$$

Introducing the *differencing operator*

$$\Delta = 1 - z^{-1}$$

and omitting the polynomial arguments, this can be written as

$$y(k) = \frac{B}{A}\, u(k-1) + \frac{1}{A\Delta}\, e(k) \tag{6.4}$$

Predicting Future Outputs

The predictive control law requires i-step-ahead predictions $\hat{y}(k+i|k)$ of the plant output as a function of future control inputs. Whereas the future values of the control input $u(k)$ are decision variables, the future values of $e(k)$ are unknown. To separate the effects of past and future noise, we write

$$\frac{1}{A\Delta} = E_i + z^{-i}\, \frac{F_i}{A\Delta} \tag{6.5}$$

where $E_i(z^{-1})$ and $F_i(z^{-1})$ are polynomials that satisfy the equation for a given $A(z^{-1})$ and prediction interval i, and the degree of $E_i(z^{-1})$ is $i-1$. The above can then be interpreted as polynomial division where F_i is the reminder; as an example, consider

$$\frac{1}{2-z^{-1}} \;=\; 1 + 2z^{-1} + \frac{4z^{-2}}{1-2z^{-1}} \;=\; E_2 + z^{-2}\frac{F_2}{1-2z^{-1}}$$

where $i = 2$, $E_2 = 1 + 2z^{-1}$ and $F_2 = 4$.

Multiplying (6.5) by $A\Delta$ yields

$$E_i A\Delta + z^{-i} F_i = 1 \tag{6.6}$$

This is a special case of a *Diophantine equation*, which has the form

$$A(z^{-1})X(z^{-1}) + B(z^{-1})Y(z^{-1}) = C(z^{-1})$$

where the polynomials A, B and C are given and the polynomials X and Y are unknown. There are infinitely many solutions X, Y to this equation: if (X_0,Y_0) is a solution pair, then clearly so is $(X_0 + BQ, Y_0 - AQ)$, where Q is an arbitrary polynomial. However if we impose the constraint that the degree of X is less than that of B, the solution is unique. Referring to (6.6), if we impose the constraint that the degree of $E_i(z^{-1})$ is $i-1$, there is a unique solution and the degree of $F_i(z^{-1})$ is less than the degree of $A\Delta$.

The special case of a Diophantine equation - as in (6.6) - where $C(z^{-1}) = 1$ is called a *Bezout identity*.

The Bezout identity (6.6) can be used to derive an i-step ahead predictor as follows. Multiply both sides by y_{k+i} to get

$$y_{k+i} = E_i A\Delta y_{k+i} + F_i y_k$$

Substitute from (6.4) for $A\Delta$ to obtain

$$y_{k+i} = \underbrace{E_i B\Delta u_{k+i-1}}_{\textit{past and future controls}} + \underbrace{F_i\, y_k}_{\text{free response}} + \underbrace{E_i\, e_{k+i}}_{\textit{future noise}} \tag{6.7}$$

In this form, the future output value y_{k+i} has been broken up into three components: one depending on past and future control inputs, one depending only on the output at sampling instant k (the "free response"), and one depending on future noise. Since $e(k)$ is zero mean white noise, the best estimate at time k is

$$\hat{y}(k+i|k) = E_i B\Delta u_{k+i-1} + F_i y_k$$

or

$$\hat{y}(k+i|k) = G_i \Delta u_{k+i-1} + F_i y_k \tag{6.8}$$

where we introduced the polynomial

$$G_i(z^{-1}) = E_i(z^{-1})B(z^{-1}) = g_0 + g_1 z^{-1} + \ldots + g_{n_g} z^{-n_g}$$

Note that the first i coefficients of G_i are the first i values of the step response of the system (6.4) when $e(k) = 0$. To see this, recall that the step response is

$$\frac{B}{A} \cdot \frac{1}{1 - z^{-1}} = \frac{B}{A\Delta} = B\left(E_i + z^{-i}\frac{F_i}{A\Delta}\right) = BE_i + z^{-i}B\frac{F_i}{A\Delta}$$

and observe that the second term on the right hand side represents only values of the step response that are delayed by at least i steps.

Optimal Control Input

The predicted outputs are needed at time k to find the control sequence that minimizes the cost (6.1). The term $G_i\Delta u_{k+i-1}$ in (6.8) depends on both past and future control values. For $i = 1$ we have

$$G_1\Delta u_k = g_0^1\Delta u_k + g_1^1\Delta u_{k-1} + g_2^1\Delta u_{k-2} + \ldots + g_{n_g}^1\Delta u_{k-n_g}$$

where only the first term on the right hand side involves future controls. For $i = 2$

$$G_2\Delta u_{k+1} = g_0^2\Delta u_{k+1} + g_1^2\Delta u_k + g_2^2\Delta u_{k-1} + \ldots + g_{n_g}^2\Delta u_{k-n_g+1}$$

the first two terms involve future controls, etc. Note that we use the notation g_l^j for the l^{th} coefficient of G_j. However, if i is the prediction interval, the first i coefficients are the values of the step response - they are identical for all G_j. If we let f_{k+i} denote the component of y_{k+i} that is known at time k, we can therefore write

$$\begin{aligned} f_{k+1} &= (G_1 - g_0)\Delta u_k + F_1 y_k \\ f_{k+2} &= (G_2 - g_0 - g_1 z^{-1})\Delta u_{k+1} + F_2 y_k \\ &\vdots \end{aligned}$$

All predicted outputs (6.8) within the prediction horizon are then determined by

$$\begin{bmatrix} \hat{y}_{k+1} \\ \hat{y}_{k+2} \\ \vdots \\ \hat{y}_{k+N_u} \\ \vdots \\ \hat{y}_{k+N_2} \end{bmatrix} = \begin{bmatrix} g_0 & 0 & \ldots & 0 \\ g_1 & g_0 & \ldots & 0 \\ \vdots & & \ddots & \vdots \\ g_{N_u-1} & g_{N_u-2} & \cdots & g_0 \\ \vdots & & & \vdots \\ g_{N_2-1} & g_{N_2-2} & \cdots & g_{N_2-N_u} \end{bmatrix} \begin{bmatrix} \Delta u_k \\ \Delta u_{k+1} \\ \vdots \\ \Delta u_{k+N_u-1} \end{bmatrix} + \begin{bmatrix} f_{k+1} \\ f_{k+2} \\ \vdots \\ f_{k+N_u} \\ \vdots \\ f_{k+N_2} \end{bmatrix} \tag{6.9}$$

where we assumed $N_1 = 1$ and $N_u < N_2$. Introducing the notation $\hat{Y}$, $\tilde{U}$, and Φ for the signal vectors and $\Gamma \in \mathbb{R}^{N_2 \times N_u}$ for the impulse response matrix in (6.9), we have

$$\hat{Y} = \Gamma\tilde{U} + \Phi$$

Introducing also a reference input vector

$$R = [r_{k+1} \;\; r_{k+2} \;\; \ldots \;\; r_{k+N_2}]^T$$

the cost function (6.1) can be written as

$$J_k = (R - \hat{Y})^T(R - \hat{Y}) + \rho\tilde{U}^T\tilde{U} = (R - \Gamma\tilde{U} - \Phi)^T(R - \Gamma\tilde{U} - \Phi) + \rho\tilde{U}^T\tilde{U}$$

Setting the derivative $dJ_k/d\tilde{U}$ to zero and solving for $\tilde{U}$ yields the minimizing vector of controls

$$\tilde{U}^* = (\Gamma^T\Gamma + \rho I)^{-1}\Gamma^T(R - \Phi) \tag{6.10}$$

Implementation

For a receding horizon strategy only the first entry of $\tilde{U}$ is needed. If we let g^T denote the first row of the matrix $(\Gamma^T\Gamma + \rho I)^{-1}\Gamma^T$, the control input to be applied at time k is computed from

$$u_k = u_{k-1} + g^T(R - \Phi) \tag{6.11}$$

If the plant model (6.4) is time-invariant, one can solve the Bezout identity (6.6) for E_i and F_i, $i = 1, \ldots, N_2$, and pre-compute g off-line. If the plant model is however time-varying, the following calculations are required in each sampling interval:

- solve (6.6) for E_i and F_i (for $i = 1, \ldots, N_2$, this can be done recursively), compute G_i
- compute Φ
- compute g
- compute $\Delta u_k = g^T(R - \Phi)$

Integral Action

The form of the control law (6.11) suggests that the controller includes integral action. To verify that error-free tracking is indeed achieved in steady state, assume that at time k all signals are constant, i.e.

$$\begin{aligned} y_k &= y_{k+1} = y_{k+2} = \ldots = \bar{y} \\ u_k &= u_{k+1} = u_{k+2} = \ldots = \bar{u} \\ r_k &= r_{k+1} = r_{k+2} = \ldots = \bar{r} \end{aligned}$$

This implies

$$\tilde{U} = 0 = (\Gamma^T\Gamma + \rho I)^{-1}\Gamma^T(\bar{R} - \bar{\Phi})$$

and therefore

$$\bar{R} = \bar{\Phi} \qquad \Rightarrow \qquad \bar{f} = \bar{r}$$

On the other hand, in steady state the Bezout identity (6.6) becomes

$$E_i(1)A(1)\Delta(1) + F_i(1) = 1$$

and because $\Delta(1) = 0$, we have

$$F_i(1) = 1$$

and thus

$$f_{k+i} = F_i y_k = y_k \qquad \Rightarrow \qquad \bar{f} = \bar{y} \qquad \Rightarrow \qquad \bar{y} = \bar{r}$$

Nonlinear Plant Model - Approximate Predictive Control

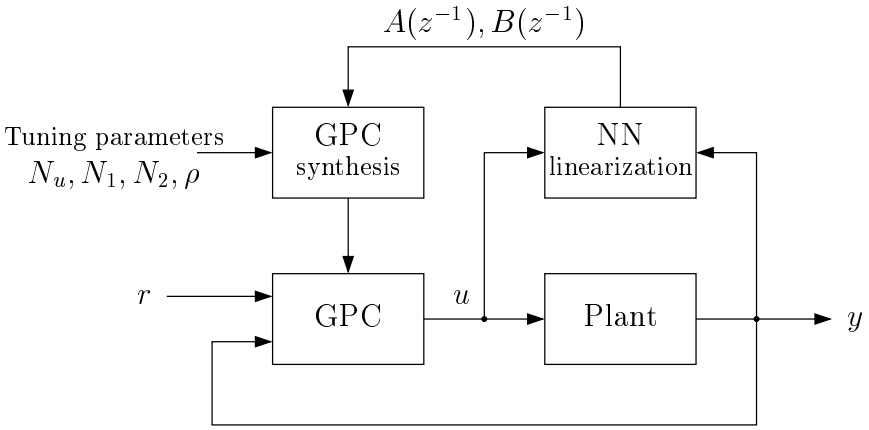

Figure 6.3: Approximate predictive control

The predictive control law discussed in the previous section assumes a - possibly time-varying - linear plant model. If a nonlinear plant is to be controlled and an MLP network model of the plant is available, then it is possible to extract a linearized plant model from the network *at each sampling instant*, and use the GPC approach based on these linearized models. This way of modelling is called *instantaneous linearization*, and the resulting control law is referred to as *approximate predictive control* (APC). The term “approximate" indicates that the optimization of the control input does not use the available nonlinear model directly when predicting outputs, but a linear approximation instead. Algorithms for on-line nonlinear optimization that use the neural network model directly have also been proposed; they use a Levenberg-Marquardt approach similar to that used for training a network. This approach is however computationally expensive, and because the current control input is computed iteratively, there is no guarantee that the algorithm converges to a solution within a sampling interval. On the other hand, the APC approach does not involve iteration, and upper bounds on the computation time can be given.

The idea of APC is shown in Figure 6.3. The bottom half shows the feedback loop with plant and GPC controller. A neural network that has been trained to capture the nonlinear dynamic behaviour of the plant is provided on-line with plant input and output data, and at each sampling instant a linearized ARX or ARMAX model is extracted and passed to the GPC synthesis block, which solves the required Bezout identities and computes the next control input. Here we will discuss only the ARX model structure.

Instantaneous Linearization

We will now derive a linearized ARX model from a nonlinear NNARX model. Thus,

consider an MLP network that represents the nonlinear function

$$\hat{y}_k = g(\varphi_k) \tag{6.12}$$

To simplify the discussion, assume initially that

$$\hat{y}_k = g(y_{k-1}, u_{k-1})$$

At sampling instant k, we wish to linearize the model about

$$\varphi_k = \begin{bmatrix} y_{k-1} \\ u_{k-1} \end{bmatrix}$$

Developing the model into a Taylor series gives

$$\hat{y}_l = y_k + \left.\frac{\partial g(\varphi_l)}{\partial y_{l-1}}\right|_{\varphi_l=\varphi_k} (y_{l-1} - y_{k-1}) + \left.\frac{\partial g(\varphi_l)}{\partial u_{l-1}}\right|_{\varphi_l=\varphi_k} (u_{l-1} - u_{k-1})$$
$$+ \text{higher order terms}$$

A linear approximation is

$$\begin{aligned} \hat{y}_l &\approx y_k + \left.\frac{\partial g(\varphi_l)}{\partial y_{l-1}}\right|_{\varphi_l=\varphi_k} (y_{l-1} - y_{k-1}) + \left.\frac{\partial g(\varphi_l)}{\partial u_{l-1}}\right|_{\varphi_l=\varphi_k} (u_{l-1} - u_{k-1}) \\ &= y_k - a_1(y_{l-1} - y_{k-1}) + b_0(u_{l-1} - u_{k-1}) \end{aligned}$$

Rearranging yields the linear model

$$\hat{y}_l = -a_1 y_{l-1} + b_0 u_{l-1} + \zeta_k$$

which has the form of an ARX model with an offset term

$$\zeta_k = y_k + a_1 y_{k-1} - b_0 u_{k-1}$$

as in (6.2).

Applying this idea to the model (6.12) with

$$\varphi_k = [y_{k-1} \ \ldots \ y_{k-n} \ \ u_{k-d} \ \ldots \ u_{k-d-m}]^T$$

yields a linear model

$$\hat{y}_k = (1 - A(z^{-1}))y_k + B(z^{-1})u_k + \zeta_k \tag{6.13}$$

where

$$\begin{aligned} A(z^{-1}) &= 1 + a_1 z^{-1} + \ldots + a_n z^{-n} \\ B(z^{-1}) &= b_0 + b_1 z^{-1} + \ldots + b_m z^{-m} \end{aligned}$$

The polynomial coefficients are given by

$$a_j = -\left.\frac{\partial g(\varphi_l)}{\partial y_{l-j}}\right|_{\varphi_l=\varphi_k}, \qquad j = 1, \ldots, n \tag{6.14}$$

$$b_j = \left.\frac{\partial g(\varphi_l)}{\partial u_{l-j}}\right|_{\varphi_l=\varphi_k}, \qquad j = 0, \ldots, m \tag{6.15}$$

and the offset term is

$$\zeta_k = A(z^{-1})y_k - B(z^{-1})u_{k-d} \tag{6.16}$$

Linearizing a Two-Layer Sig-Lin Perceptron Network

From (6.12), the partial derivatives in (6.14) and (6.15) are obtained by computing

$$\left.\frac{\partial \hat{y}(l)}{\partial \varphi_i(l)}\right|_{l=k}$$

For a two-layer sig-lin perceptron network, the output at time l is

$$\hat{y}_l = \sum_{j=1}^{s^1} w_{1j}^2 \tanh\left(\sum_{i=1}^{r} w_{ji}^1 \varphi_i(l) + w_{j0}^1\right) + w_0^2$$

and applying the chain rule yields

$$\left.\frac{\partial \hat{y}(l)}{\partial \varphi_i(l)}\right|_{l=k} = \sum_{j=1}^{s^1} w_{1j}^2 w_{ji}^1 \left[1 - \tanh^2\left(\sum_{i=1}^{r} w_{ji}^1 \varphi_i(l) + w_{j0}^1\right)\right] \tag{6.17}$$

Computing the right hand side gives $-a_1, \ldots, -a_n$ when $i = 1, \ldots, n$ and $b_0, \ldots, b_m$ when $i = n+1, \ldots, n+m+1$.

Exercises

Problem 6.1

Train a neural network to capture the behaviour of the CSTR pH process given in Exercise 3.2, and design an approximate predictive controller based on instantaneous linearization. Use the MATLAB tools provided in the NNSYSID and NNCONTROL toolbox (a link to the download site is provided on the web page of this course). Tune the controller for the reference trajectory given in Exercise 3.2, and compare the achievable performance and the design effort with that of the PID controller.

Chapter 7

Linear Subspace Identification

The discussion in the previous chapters was limited to SISO transfer function models. The approach presented there can be extended to cover MIMO models , but working with multivariable systems is usually more convenient in a state space framework. In this and the following section we present a recently developed approach to estimating linear SISO and MIMO state space models.

To introduce the idea, we begin with a SISO state space model

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k), \qquad x(0) = 0 \\ y(k) &= cx(k) + du(k) \end{aligned}$$

(Note that we use the same symbol Γ in discrete-time SISO and MIMO models.) Now assume that $x(0) = 0$, and consider the impulse response of the above model, i.e. the response to the input $u(k) = \delta(k)$. Observing that for $k > 0$ we have $x(k) = \Phi^{k-1}\Gamma$, we find that the impulse response $g(k)$ is given by

$$g(k) = \begin{cases} 0, & k < 0 \\ d, & k = 0 \\ c\Phi^{k-1}\Gamma, & k > 0 \end{cases}$$

The values $\{d,\ c\Gamma,\ c\Phi\Gamma, \ldots\}$ of the impulse response sequence are called the *Markov parameters* of the system.

Turning now to multivariable systems, we first need to clarify what we mean by the impulse response of a MIMO model

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma u(k), \qquad x(0) = 0 \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{7.1}$$

We can apply a unit impulse to one input channel at a time and observe the resulting

response at each output channel

$$u_{\delta i}(k) = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \delta(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \rightarrow \quad y_{\delta i}(k) = \begin{bmatrix} g_{1i}(k) \\ \vdots \\ g_{li}(k) \end{bmatrix}$$

Here $\delta(k)$ is placed in the i^{th} entry of the input vector, while all other inputs are zero. An entry $g_{ji}(k)$ in the output vector represents the response at output channel j to a unit impulse applied at input channel i. The complete information about the impulse responses from each input to each output can then be represented by the impulse response matrix

$$g(k) = \begin{bmatrix} g_{11}(k) & \dots & g_{1m}(k) \\ \vdots & & \vdots \\ g_{l1}(k) & \dots & g_{lm}(k) \end{bmatrix}$$

Introducing the notation

$$\Gamma = [\Gamma_1 \ \ \Gamma_2 \ \dots \ \Gamma_m], \qquad D = [d_1 \ \ d_2 \ \dots \ d_m]$$

where Γ_i and d_i denote the i^{th} column of the matrices Γ and D, respectively, we find that with input $u_{\delta i}(k)$ we have $x_{\delta i}(k) = \Phi^{k-1}\Gamma_i$ for $k > 0$, and at $k = 0$ we have $y_{\delta i}(k) = d_i$. Combining the responses to impulses at all input channels, we obtain

$$g(k) = [y_{\delta 1}(k) \ \dots \ y_{\delta m}(k)] = \begin{cases} 0, & k < 0 \\ D, & k = 0 \\ C\Phi^{k-1}\Gamma, & k > 0 \end{cases} \tag{7.2}$$

The impulse response describes input-output properties of a system, and we would expect it to be independent of a particular coordinate basis that has been chosen for a given state space model. This seems to contradict the fact that the impulse response in (7.2) is given in terms of the matrices (Φ, Γ, C) of a state space model, which clearly depend on the choice of coordinate basis. However, it is easily checked that applying a similarity transformation T - which yields a realization $(T^{-1}\Phi T, T^{-1}\Gamma, CT)$ - will not change the impulse response.

Constructing a Model from the Impulse Response

Assume that measured impulse response data of a system are available and have been

arranged in the form of a matrix

$$H_k = \begin{bmatrix} g(1) & g(2) & g(3) & \dots & g(k) \\ g(2) & g(3) & \dots & & g(k+1) \\ g(3) & \vdots & & & \\ \vdots & & & & \vdots \\ g(k) & g(k+1) & & \dots & g(2k-1) \end{bmatrix}$$

A matrix with this structure is called a *Hankel matrix* if the $g(l)$ are scalar, and a *block-Hankel matrix* if the $g(l)$ are matrices. Using (7.2) in the above we obtain

$$H_k = \begin{bmatrix} C\Gamma & C\Phi\Gamma & C\Phi^2\Gamma & \dots & C\Phi^{k-1}\Gamma \\ C\Phi\Gamma & C\Phi^2\Gamma & \dots & & C\Phi^k\Gamma \\ C\Phi^2\Gamma & \vdots & & & \\ \vdots & & & & \vdots \\ C\Phi^{k-1}\Gamma & C\Phi^k\Gamma & & \dots & C\Phi^{2k-2}\Gamma \end{bmatrix} \tag{7.3}$$

Assume the number of samples is sufficiently large so that $k > n$, where n is the order of the state space model. Note that at this point we know nothing about the system apart from its impulse response. In particular, we do not know the order n of the system. Important in this context is the rank of the matrix H_k. To investigate this, we first observe that we can factor H_k as

$$H_k = \begin{bmatrix} C \\ C\Phi \\ \vdots \\ C\Phi^{k-1} \end{bmatrix} [\Gamma \;\; \Phi\Gamma \;\; \dots \Phi^{k-1}\Gamma] = \mathcal{O}_k \mathcal{C}_k$$

Here $\mathcal{O}_k$ and $\mathcal{C}_k$ are the *extended observability* and *controllability matrices*, respectively, of the model (7.1), where the term "extended" is added because the number of samples k is greater than the expected order n of the system. Assuming that we are interested in estimating a model (7.1) that represents a minimal realization of a system, i.e. if (Φ, Γ) is controllable and (C, Φ) is observable, then we have

$$\text{rank } \mathcal{O}_k = \text{rank } \mathcal{C}_k = n$$

which implies

$$\text{rank } H_k = n \tag{7.4}$$

Thus, we can obtain the order from the measured data by computing the rank of H_k.

The Ideal Case

Assume that for a given system with m inputs and l outputs the measured discrete-time impulse response $g(k)$ is available, and that we want to identify a discrete-time state

space model (Φ, Γ, C). For systems where $D \neq 0$, the feedthrough matrix is given by $g(0)$. Initially we do not know the dynamic order of this model, but we assume that we have a sufficient number of samples of the impulse response, so that we can form the $mk \times lk$ matrix H_k for a value of k that is larger than the expected order of the model. If the impulse response data were indeed generated by a linear state space model of the form (7.1) with n state variables, and if no measurement errors are present in the data, then the order n of the model can be easily determined by checking the rank of H_k. Knowing n we can then factor H_k as

$$H_k = ML, \qquad M \in \mathbb{R}^{lk \times n},\ L \in \mathbb{R}^{n \times mk}$$

such that

$$\text{rank } M = \text{rank } L = n$$

This can be done using singular value decomposition as explained below. Note that this factorization is not unique. Finally, we define the matrices M and L to be the *extended observability and controllability matrices*

$$\mathcal{O}_k = M, \qquad \mathcal{C}_k = L$$

The first l rows of M therefore represent the measurement matrix C, and the first m columns of L form the input matrix Γ. To find the state matrix Φ, define

$$\bar{\mathcal{O}}_k = \begin{bmatrix} C\Phi \\ \vdots \\ C\Phi^k \end{bmatrix} = \mathcal{O}_k \Phi$$

Note that we can generate $\bar{\mathcal{O}}_k$ from measured data by factorizing the larger Hankel matrix H_{k+1} and removing the first l rows from $\mathcal{O}_{k+1}$. Multiplying the above from the left by $\mathcal{O}_k^T$ we obtain

$$\mathcal{O}_k^T \mathcal{O}_k \Phi = \mathcal{O}_k^T \bar{\mathcal{O}}_k$$

Since $\mathcal{O}_k$ has full row rank, we can compute Φ from

$$\Phi = (\mathcal{O}_k^T \mathcal{O}_k)^{-1} \mathcal{O}_k^T \bar{\mathcal{O}}_k$$

The above describes a procedure for constructing the matrices Φ, Γ and C from measured impulse response data. At this point a question arises: we know that a state space model of a given system is not unique but depends on the coordinate basis chosen for the state space. One could therefore ask where this choice was made in the above construction. The answer is that the factorization $H_k = ML$ is not unique, in fact if M and L are factors of rank n and if T is an arbitrary nonsingular $n \times n$ matrix, then it is easy to see that MT and $T^{-1}L$ are also rank n factors of H_k. With this latter choice we obtain

$$\tilde{\mathcal{O}}_k = \mathcal{O}_k T, \qquad \tilde{\mathcal{C}}_k = T^{-1} \mathcal{C}_k$$

We know however that these are the observability and controllability matrices, respectively, of the model obtained by applying the similarity transformation T to (Φ, Γ, C). This shows that a choice of coordinate basis is made implicitly when H_k is factored into ML.

Modelling Errors and Measurement Noise

The above procedure for identifying a state space model relies on the assumption that the measured data were indeed generated by a linear system and are not corrupted by measurement noise. In practice, neither assumption will be true. One consequence of this is that no matter how large k is chosen, the matrix H_k will usually have full rank. To extract information about the model order in spite of data being corrupted, one can use the technique of *singular value decomposition*, a brief review of which is given in the Appendix.

Singular Value Decomposition

Consider a singular value decomposition of the data matrix $H_k \in \mathbb{R}^{kl \times km}$

$$H_k = Q \Sigma V^T \tag{7.5}$$

where Σ is a diagonal matrix with nonnegative diagonal entries, and Q and V are orthogonal, i.e. they satisfy

$$QQ^T = I_{kl}, \qquad VV^T = I_{km}$$

Assume that the singular values are arranged in decreasing order such that

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots \sigma_p = 0$$

where $p = \min(kl, km)$. In this case we have rank $H_k = r$, because an important fact about singular value decomposition is that the rank of a matrix is equal to the number of its nonzero singular values. Here σ_{r+1} and all the following singular values are zero. On the other hand, if the singular values $\sigma_{r+1}, \ldots, \sigma_p$ are very small - much smaller than σ_r - but nonzero, the matrix H_k has full rank but is "close to being singular". This is precisely the situation we encounter when a block-Hankel matrix is constructed from impulse response data that are corrupted by measurement noise. One of the powerful features of the singular value decomposition is that it allows us to distinguish significant information from noise effects by inspection of the singular values. An example is shown in Fig. 7.1, where the singular values of a matrix are shown in decreasing order. If these were the singular values of a Hankel matrix constructed from a measured impulse response, we would conclude that the system dynamics can be described reasonably well by a 4^{th} order model and that the remaining nonzero but small singular values represent noise effects.

If σ_{r+1} is much smaller than σ_r, we say that the *numerical rank* of H_k is r. Another way

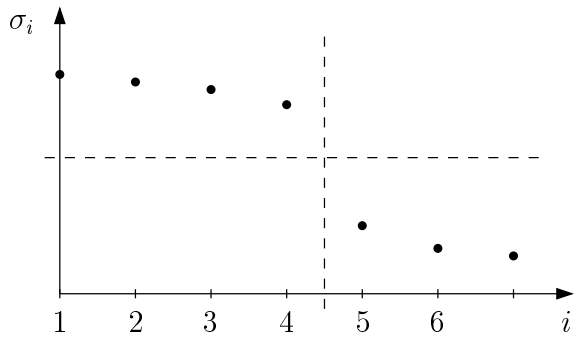

Figure 7.1: Determining the numerical rank

of looking at this is to write (7.5) as

$$H_k = \begin{bmatrix} & & \\ q_1 & q_2 & \dots & q_{kl} \\ & & \end{bmatrix} \begin{bmatrix} \sigma_1 & & & 0 & 0 & \dots & 0 \\ & \ddots & & & 0 & \dots & 0 \\ 0 & & \sigma_p & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} & v_1^T & \\ & v_2^T & \\ & \vdots & \\ & v_{km}^T & \end{bmatrix}$$

where q_i and v_i represent the i^{th} column of Q and V, respectively. Expanding the right hand side column by column, we obtain

$$H_k = \sum_{i=1}^{p} \sigma_i q_i v_i^T = \sum_{i=1}^{r} \sigma_i q_i v_i^T + \sum_{i=r+1}^{p} \sigma_i q_i v_i^T = Q_s \Sigma_s V_s^T + Q_n \Sigma_n V_n^T$$

where $Q_s \in \mathbb{R}^{kl \times r}$ and $V_s \in \mathbb{R}^{km \times r}$ are the matrices formed by the first r columns of Q and V, respectively. The matrices $Q_n \in \mathbb{R}^{kl \times (kl-r)}$ and $V_n \in \mathbb{R}^{km \times (km-r)}$ are similarly formed by the remaining columns. If the singular values $\sigma_{r+1}, \dots, \sigma_p$ are much smaller than σ_r, the last term on the right hand side can be neglected and we have

$$H_k \approx Q_s \Sigma_s V_s^T$$

or

$$H_k \approx \begin{bmatrix} & & \\ q_1 & q_2 & \dots & q_r \\ & & \end{bmatrix} \begin{bmatrix} \sigma_1 & & & 0 & 0 & \dots & 0 \\ & \ddots & & & 0 & \dots & 0 \\ 0 & & \sigma_r & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} & v_1^T & \\ & v_2^T & \\ & \vdots & \\ & v_r^T & \end{bmatrix} = Q_s \Sigma_s^{1/2} \Sigma_s^{1/2} V_s^T$$

where r is the numerical rank of H_k. Now taking r as the estimated model order $\hat{n}$, we can define the extended observability and controllability matrices $\mathcal{O}_r \in \mathbb{R}^{kl \times r}$ and $\mathcal{C}_r \in \mathbb{R}^{km \times r}$ as

$$\mathcal{O}_r = Q_s \Sigma_s^{1/2}, \qquad \mathcal{C}_r = \Sigma_s^{1/2} V_s^T$$

A state space model (Φ, Γ, C) of order $\hat{n}$ can then be obtained as in the case of ideal measurements.

The identification of a state space model from the impulse response is illustrated in Exercise 7.1.

Direct Subspace Identification

The method outlined in the previous section assumes that the measured impulse response is available. In practice it is usually better to use more general data, obtained for example by applying a white noise input signal. We will now present a technique for identifying state space models without using the measured impulse response, referred to as *direct subspace identification*.

Consider again the model (7.1). Beginning at time k, the output at successive time instants is given by

$$\begin{aligned} y(k) &= Cx(k) + Du(k) \\ y(k+1) &= C\Phi x(k) + C\Gamma u(k) + Du(k+1) \\ y(k+2) &= C\Phi^2 x(k) + C\Phi\Gamma u(k) + C\Gamma u(k+1) + Du(k+2) \\ &\vdots \end{aligned}$$

Introducing the vectors

$$Y_k = \begin{bmatrix} y(k) \\ y(k+1) \\ \vdots \\ y(k+\alpha-1) \end{bmatrix}, \qquad U_k = \begin{bmatrix} u(k) \\ u(k+1) \\ \vdots \\ u(k+\alpha-1) \end{bmatrix}$$

of input and output data, we can write

$$Y_k = \mathcal{O}_\alpha x(k) + \Psi_\alpha U_k \tag{7.6}$$

where

$$\mathcal{O}_\alpha = \begin{bmatrix} C \\ C\Phi \\ C\Phi^2 \\ \vdots \\ C\Phi^{\alpha-1} \end{bmatrix}, \qquad \Psi_\alpha = \begin{bmatrix} D & 0 & 0 & \dots & 0 \\ C\Gamma & D & 0 & \dots & 0 \\ C\Phi\Gamma & C\Gamma & D & & 0 \\ \vdots & & \ddots & \ddots & \\ C\Phi^{\alpha-2}\Gamma & C\Phi^{\alpha-1}\Gamma & \dots & C\Gamma & D \end{bmatrix}$$

Assume that a sufficient number of measurements has been collected so that we can form the input and output data matrices

$$\mathcal{Y} = [Y_1 \;\; Y_2 \;\; \dots \;\; Y_N], \qquad \mathcal{U} = [U_1 \;\; U_2 \;\; \dots \;\; U_N],$$

Define also the matrix of state variables

$$X = [x(1) \;\; x(2) \;\; \dots \;\; x(N)]$$

From (7.6), these data matrices satisfy

$$\mathcal{Y} = \mathcal{O}_\alpha X + \Psi_\alpha \mathcal{U} \tag{7.7}$$

In this equation only $\mathcal{U}$ and $\mathcal{Y}$ are known; note that $\mathcal{U} \in \mathbb{R}^{m\alpha \times N}$. We assume that the number $(N + \alpha - 1)$ of measurements - which is required to fill the above matrices - is large enough such that α can be chosen greater than the expected model order, and N such that $N > m\alpha$. To identify a state space model, we need to estimate the matrices $\mathcal{O}_\alpha$ (from which C and Φ can be extracted) and Ψ_α (from which we get D and Γ).

Estimating the Term $\mathcal{O}_\alpha X$

As a first step, we will eliminate the effect of the input data on the output data in (7.7) and estimate the product $\mathcal{O}_\alpha X$. This can be achieved by projecting the output data onto the *nullspace* of the input data matrix $\mathcal{U}$. The nullspace $\mathcal{N}(\mathcal{U})$ is defined as the space of all vectors q that are made zero when multiplied from the left by $\mathcal{U}$:

$$\mathcal{N}(\mathcal{U}) = \{q : \mathcal{U}q = 0\}$$

Now define the matrix Π as

$$\Pi = I - \mathcal{U}^T(\mathcal{U}\mathcal{U}^T)^{-1}\mathcal{U}$$

All columns of Π are orthogonal to $\mathcal{U}$, this can be seen from

$$\mathcal{U}\Pi = \mathcal{U} - \mathcal{U}\mathcal{U}^T(\mathcal{U}\mathcal{U}^T)^{-1}\mathcal{U} = 0$$

Note that Π is constructed from measured data only. Here we assumed that $(\mathcal{U}\mathcal{U}^T)$ is invertible, a condition for this is that the input is persistently exciting of order $m\alpha$. Multiplying equation (7.7) from the right by Π then yields

$$\mathcal{Y}\Pi = (\mathcal{O}_\alpha X + \Psi_\alpha \mathcal{U})\Pi = \mathcal{O}_\alpha X \Pi$$

The left hand side is known (because it is constructed from measured data), thus the product $\mathcal{O}_\alpha X\Pi$ is known. Observing that $\mathcal{O}_\alpha \in \mathbb{R}^{l\alpha \times n}$ and $X\Pi \in \mathbb{R}^{n \times N}$, we can obtain an estimate of the extended observability matrix by determining the numerical rank $\hat{n}$ of the matrix $\mathcal{Y}\Pi$ and by factoring it into a left factor with $\hat{n}$ columns and full column rank, and a right factor with $\hat{n}$ rows. This can be done by computing the singular value decomposition

$$\mathcal{Y}\Pi = Q_s \Sigma_s V_s^T + Q_n \Sigma_n V_n^T \approx Q_s \Sigma_s^{1/2} \Sigma_s^{1/2} V_s^T$$

and by taking

$$\mathcal{O}_\alpha = Q_s \Sigma_s^{1/2}$$

Here again the order of the system is estimated by inspection of the singular values - this time of the data matrix $\mathcal{Y}\Pi$. From $\mathcal{O}_\alpha$ the matrices C and Φ can be obtained as described in the previous section.

Estimating the Term $\Psi_\alpha \mathcal{U}$

We can now use the estimate of $\mathcal{O}_\alpha$ to eliminate the first term on the right hand side of (7.7). For this purpose, observe that from

$$QQ^T = \begin{bmatrix} Q_s^T \\ Q_n^T \end{bmatrix} [Q_s \;\; Q_n] = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

we have $Q_n^T Q_s = 0$ and therefore $Q_n^T \mathcal{O}_\alpha \approx 0$. Thus, from (7.7) we obtain

$$Q_n^T \mathcal{Y} \mathcal{U}^{-R} = Q_n^T \Psi_\alpha$$

where $\mathcal{U}^{-R} = \mathcal{U}^T (\mathcal{U}\mathcal{U}^T)^{-1}$ denotes the right inverse of $\mathcal{U}$. The left hand side of this equation and Q_n are known, so that Ψ_α is the only unknown term. The matrices Γ and D can then be obtained by solving a linear system of equations, details are omitted.

Exercises

Problem 7.1

Download the Matlab script `cs7_mkdata.m`. This script generates the impulse responses $g(k)$ and $gn(k)$ of a system with 2 inputs and 2 outputs. The sequence $g(k)$ is noise free, whereas $gn(k)$ is corrupted by measurement noise.

a) Generate from the sequence $g(k)$ the block Hankel matrix of the impulse response. Estimate upper and lower limits for the order n of the system and determine by factorization of the block Hankel matrix linear state space models of the system for different values of the order n. Compare the impulse responses of the estimated models with the output sequence $g(k)$.

   Hint: You can use the function `mkhankel.m` *to generate the Hankel matrix.*

b) Repeat the estimation for the noisy impulse response $gn(k)$.

Problem 7.2

This exercise uses the Matlab Identification toolbox GUI `ident` to identify a state space model from sets of data with two inputs and two outputs. The data is in the file `cs9_identGUI.mat`.

Two sets data are contained in the file,`iodata1` and `iodata2`. They are in the `Matlab` format `iddata` that can be directly imported into `ident`.

a) Import the data set `iodata1` and generate direct subspace identified models of different orders using the command `n4sid`.

b) validate the models generated against the data set `iodata2`. What is the model order that most effectively describes the plant behaviour?

Bibliography

[1] L. Ljung, *System Identification - Theory for the User*. Amsterdam: Pearson Education, 1998, ISBN: 978-0-132-44053-0.

[2] M. Norgaard, O. Ravn, N. Poulsen, and L. Hansen, *Neural Networks for Modelling and Control of Dynamic Systems - A Practitioners Handbook*. London: Springer London, 2003, ISBN: 978-1-852-33227-3.

[3] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*. Brooks/Cole, 1996, ISBN: 978-0-534-95259-4.

[4] G. Strang, *Linear Algebra and Its Applications*. Stanford: Elsevier Science, 2014, ISBN: 978-1-483-26511-7.

[5] T. Mathworks, *System identification overview - matlab & simulink*, 2016. [Online]. Available: `https://de.mathworks.com/help/ident/gs/about-system-identification.html`.

Appendix A

APPENDICES

Appendix A

Solutions to Exercises

Chapter 1

Solution to Problem q:LSident Proof of least squares optimality

We calculate the θ which makes the first derivative of $V(\theta)$ zero, to find the θ which minimizes $V(\theta)$. We have

$$\begin{aligned} V(\theta) &= (Y-\Phi\theta)^T(Y-\Phi\theta) \\ &= Y^TY - \theta^T\Phi^TY - Y^T\Phi\theta + \theta^T\Phi^T\Phi\theta \end{aligned}$$

So

$$\begin{aligned} \frac{dV(\theta)}{d\theta} &= -\Phi^TY - \Phi^TY + 2\Phi^T\Phi\theta = 0 \\ 0 &= -\Phi^TY + \Phi^T\Phi\theta \\ \theta &= (\Phi^T\Phi)^{-1}\Phi^TY \end{aligned}$$

Another way of finding the θ which minimizes $V(\theta)$ is by using the approach of completion of squares. Let

$$\begin{aligned} V(\theta) &= (Y-\Phi\theta)^T(Y-\Phi\theta) \\ &= Y^TY - \theta^T\Phi^TY - Y^T\Phi\theta + \theta^T\Phi^T\Phi\theta \end{aligned}$$

or

$$V(\theta) - Y^TY = \theta^T\Phi^T\Phi\theta - \theta^T\Phi^TY - Y^T\Phi\theta$$

Adding a constant term $Y^T\Phi(\Phi^T\Phi)^{-1}\Phi^TY$ to both side will yield

$$\begin{aligned} V(\theta) - Y^TY + Y^T\Phi(\Phi^T\Phi)^{-1}\Phi^TY &= \theta^T\Phi^T\Phi\theta - \theta^T\Phi^TY - Y^T\Phi\theta \\ &\quad + Y^T\Phi(\Phi^T\Phi)^{-1}\Phi^TY \\ &= (\theta - (\Phi^T\Phi)^{-1}\Phi^TY)^T\Phi^T\Phi(\theta - (\Phi^T\Phi)^{-1}\Phi^TY) \end{aligned}$$

Thus

$$V(\theta) = (\theta - (\Phi^T\Phi)^{-1}\Phi^T Y)^T\Phi^T\Phi(\theta - (\Phi^T\Phi)^{-1}\Phi^T Y) + Y^T Y - Y^T\Phi(\Phi^T\Phi)^{-1}\Phi^T Y$$

Which shows that $V(\theta)$ is minimum if the first term on right hand side is minimum or equal to zero. Then

$$\theta - (\Phi^T\Phi)^{-1}\Phi^T Y = 0$$

or

$$\theta = (\Phi^T\Phi)^{-1}\Phi^T Y$$

Solution to Problem q:PEcalc Persistent excitation of step functions

a)
$$(z-1)u(kT) = u(kT+T) - u(kT)$$
For the step function $u(t+T) - u(t)$ is only 1 at $k = -1$.

b) If, for **all** polynomials $a(z)$ of order n $(a_n z^n + \ldots)$:
$$\lim_{k\to\infty} \frac{1}{k}\sum_{l=0}^{k} (a(z)u(l))^2 > 0$$
the PE order is $n+1$.

This means that if **any** polynomial $a(z)$ can be found such that
$$\lim_{k\to\infty} \frac{1}{k}\sum_{l=0}^{k} (a(z)u(l))^2 = 0$$
then the PE order must be $\leq n$.

c) With the polynomial $a(z) = z - 1$ (order $n = 1$) and $u(l)$ a step function, then from a).
$$a(z)u(l) = 0, \qquad l = 0, 1, \ldots \infty$$
so
$$\lim_{k\to\infty} \frac{1}{k}\sum_{l=0}^{k} (a(z)u(l))^2 = 0$$
so the PE order is either 1 or 0.

d) Next, we will find the exact order by analyzing the auto correlation $C_{uu}(1) = c_{uu}(0)$. Since, it is a scaler hence it has rank of 1 or PE order is 1.

Solution to Problem q:PEsin Persistent excitation of sinusoid

a)

$$(z^2 - 2z\cos\omega T + 1)u(kT) = \sin(\omega kT + 2\omega T) - 2\cos\omega T \sin(\omega kT + \omega T) + \sin(\omega kT)$$

Since,

$$\sin A + \sin B = 2\sin\frac{A+B}{2}\cos\frac{A-B}{2}$$

$$\sin(\omega kT + 2\omega T) + \sin(\omega kT) = 2\sin(\omega kT + \omega T)\cos\omega T$$
$$(z^2 - 2z\cos\omega T + 1)u(kT) = 0$$

b) If we can find any polynomial $a(z)$ of order n, such that

$$\lim_{k\to\infty}\frac{1}{k}\sum_{l=0}^{k}\left(a(z)u(l)\right)^2 = 0$$

Then, the PE order must be $\leq n$

With $a(z) = (z^2 - 2z\cos\omega T + 1)$, i.e. $n = 2$. As shown in a),

$$a(z)u(l) = 0, \qquad \forall l = 0, 1, \ldots \infty$$

Then,

$$\lim_{k\to\infty}\frac{1}{k}\sum_{l=0}^{k}\left(a(z)u(l)\right)^2 = 0$$

So the PE order must be ≤ 2.

c) For the input signal $u(kT)$

$$R_u(\tau) = \lim_{N\to\infty}\frac{1}{N}\sum_{k=0}^{N}u(kT)u(kT \pm \tau)$$

At times 0 and T

$$R_u(0) = \lim_{N\to\infty}\frac{1}{N}\sum_{k=0}^{N}u(kT)u(kT)$$
$$R_u(T) = \lim_{N\to\infty}\frac{1}{N}\sum_{k=0}^{N}u(kT)u\Big((k+1)T\Big)$$

These are the elements $(1,1)$ and $(1,2)$ of $C_{uu}(2)$, so

$$C_{uu}(2) = \begin{bmatrix} R_u(0) & R_u(T) \\ R_u(T) & R_u(0) \end{bmatrix}$$
$$C_{uu}(2) = \begin{bmatrix} \frac{1}{2}\cos 0 & \frac{1}{2}\cos\omega T \\ \frac{1}{2}\cos\omega T & \frac{1}{2}\cos 0 \end{bmatrix}$$
$$C_{uu}(2) = \frac{1}{2}\begin{bmatrix} 1 & \cos\omega T \\ \cos\omega T & 1 \end{bmatrix}$$

d) At $T = \frac{2\pi}{\omega}$

$$C_{uu}(2) = \begin{bmatrix} \frac{1}{2}\cos 0 & \frac{1}{2}\cos \omega T \\ \frac{1}{2}\cos \omega T & \frac{1}{2}\cos 0 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

$$\text{rank} C_{uu}(2) = 1$$

So PE order is 1. All the samples are at the same position in the sine wave so it looks like a step.

At $\omega \neq \frac{2\pi}{T}$, the samples are at different positions in the sine wave, so it has more information: PE order =2.

Solution to Problem q:PEwn Persistent excitation of white noise

For white noise,

$$C_{uu}(1) = c_{uu}(0) = \lim_{k\to\infty} \frac{1}{k} \sum_{i=0}^{k} u_i^2 = S_0$$

$$C_{uu}(2) = \begin{bmatrix} c_{uu}(0) & c_{uu}(1) \\ c_{uu}(1) & c_{uu}(0) \end{bmatrix}$$

Using, the property of white noise that $c_{uu}(i) = 0, \forall i = 1, 2 \ldots,$

$$\begin{aligned} C_{uu}(2) &= \begin{bmatrix} c_{uu}(0) & 0 \\ 0 & c_{uu}(0) \end{bmatrix} \\ &= S_0 I_2 \\ &\vdots \\ C_{uu}(n) &= S_0 I_n \end{aligned}$$

so $C_{uu}(n)$ has rank n or PE condition is satisfied for all n.

Solution to Problem q:identex Least Squares identification

a) Let us have N samples and let $n = 2$ then,

$$\begin{aligned} y_0 &= -a_1 y_{-1} - a_2 y_{-2} + b_1 u_{-1} + b_2 u_{-2} + e_0 \\ y_1 &= -a_1 y_0 - a_2 y_{-1} + b_1 u_0 + b_2 u_{-1} + e_1 \\ y_2 &= -a_1 y_1 - a_2 y_0 + b_1 u_1 + b_2 u_0 + e_2 \\ y_3 &= -a_1 y_2 - a_2 y_1 + b_1 u_2 + b_2 u_1 + e_3 \\ &\vdots \\ y_{N-1} &= -a_1 y_{N-2} - a_2 y_{N-3} + b_1 u_{N-2} + b_2 u_{N-3} + e_{N-1} \end{aligned}$$

or

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} -y_{-1} & -y_{-2} & u_{-1} & u_{-2} \\ -y_0 & -y_{-1} & u_0 & u_{-1} \\ \vdots & \vdots & \vdots & \vdots \\ -y_{N-2} & -y_{N-3} & u_{N-2} & u_{N-3} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ b_1 \\ b_2 \end{bmatrix} + \begin{bmatrix} e_2 \\ e_3 \\ \vdots \\ e_{N-1} \end{bmatrix}$$

However, the values of input sequence $u(-1)$, $u(-2)$, $...u(-n)$, and $y(-1)$, $y(-2)$, $...y(-n)$ is not available in measurement data. This means that first n-rows will be 0, hence to make $\Phi^T\Phi$ full rank these rows should be eliminated. This results in,

$$\Phi = \begin{bmatrix} -y_1 & -y_0 & u_1 & u_0 \\ -y_2 & -y_1 & u_2 & u_1 \\ \vdots & \vdots & \vdots & \vdots \\ -y_{N-2} & -y_{N-3} & u_{N-2} & u_{N-3} \end{bmatrix}$$

which has the dimensions of $N - n \times 2n$:

b) See `cs7_LSrankM.m`.

For the sinusoid: rank = 4 (singular values confirm this).

For the white noise: rank = arbitrary

As the sequence becomes longer, the matrix $\Phi^T\Phi$ approaches a scaled version of the empirical covariance matrix; thus the rank of $\Phi^T\Phi$ for a long sequence can be expected to have the same rank as the PE order.

c) See Matlab solution in `cs7_LSparest.m`. 3rd and 4th order models generated are identical. A pole and zero cancel in the 4th order model.

d) Exact validation achieved with these models: the model order is clearly 3.

e) Inconsistent results when attempt to generate models from sinusoidal or step input. A true inverse is only possible when rank $\Phi^T\Phi = 2n$: with a PE order of 2 it is only possible to accurately estimate a system of order 1 (which has 2 parameters).

Chapter 3

Solution to Problem q:mlp Single-Input MLP Network

a)

$$y = \frac{w_{11}^2}{1 + e^{-(\varphi w_{11}^1 + w_{10}^1)}} + \frac{w_{12}^2}{1 + e^{-(\varphi w_{21}^1 + w_{20}^1)}} + w_{10}^2$$

The first inflection point is at $(-7, 0)$ and the second at $(5, 3)$. For the steepness the relation of $w_1^{'}$ and w_1 as seen in Figure 3.13 can be used. One solution would be

$$w_{11}^2 = -2, w_{12}^2 = 8, w_{10}^2 = 1, w_{11}^1 = 3, w_{10}^1 = 21, w_{21}^1 = 0.5, w_{20}^1 = -2.5.$$

For using the *nntool* from MATLAB to verify the weights, follow these steps:

- Type *nntool* in the MATLAB command window.
- Import the vector φ as *Input Data* and the vector *g_phi* as *Target Data*.
- Click *New* and choose *Feed-forward backprop* as network type and φ and *g_phi* as the input and target data, respectively.
- Click on your network in the *Networks* field of the *Network/Data Manager*. *View* the network. Set the weights as chosen above.
- Choose *Simulate*, specify φ as your input and simulate the network.
- Export the output to the workspace and compare it with the given function .

b)
- Type *nnstart* in the MATLAB command window.
- Start the *Fitting app*.
- Import the vector φ as *Input Data* and the vector *g_phi* as *Target Data*.
- Choose the appropriate *Number of Hidden Neurons*.
- *Train* the network.
- Click on *Plot Fit*. Retrain to reduce the error between the *Target* and *Output* is small.

Chapter 7

Solution to Problem q:subspaceID Subspace identification

a) Clear cut-off, 9 non-zero singular values in Hankel matrix. Reasonable model achieved with model order 4. The model can be estimated using `cs7_parest.m`

b) The cut-off is not so clear for the noisy signal. Since after the 4th singular value of `Hn` the others are relative small one can chose 4th or 5th order model. Because the difference between the 3rd and the 4th singular values is also large one can try also identifying a 3rd order model of the system.

Solution to Problem q:IDtoolbox Subspace identification using ident GUI

1. Open the toolbox with the command `ident`.
2. Import the data: *Import data* → *Data object* → *Object*: `iodata1`. Repeat for the second data set.
3. The signals then appear in the left panel *Data Views*.
4. Drag and drop the first signal set to *Working model*. Remove the means from all signals using the *Preprocess* drop-down list → *Remove means*. Repeat for the second signal set. One of the new set of signals should be used as *Working data* and the other one as *Validation data*.
5. Estimate models of 2nd, 3rd, 4th and 5th order using N4SID (subspace identification). For the purpose choose *Linear parametric models* from the *Estimate* drop-down box. Select *State-space* as *Structure* and repeat the identification for the different orders.
6. Validate the identified models using the second data set. Use the *Model Views* check-boxes in the lower-right corner of the GUI.

The model used to create the data was 4th order with noise. The identified models of order 2-4 all very accurately reproduce the original data.