Nonlinear empirical modeling using local PLS models

Lars Aarhus

August 17, 1994

Abstract

This thesis proposes some new iterative local modeling algorithms for the multivariate approximation problem (mapping from $\mathcal{R}^P$ to $\mathcal{R}$). Partial Least Squares Regression (PLS) is used as the local linear modeling technique. The local models are interpolated by means of normalized Gaussian weight functions, providing a smooth total nonlinear model. The algorithms are tested on both artificial and real world set of data, yielding good predictions compared to other linear and nonlinear techniques.

Preface

This thesis[a] completes my work for the degree Candidatus Scientarum at the Department of Informatics, University of Oslo. It has been carried out at SINTEF, Oslo during two years from August 1992 to August 1994. My supervisors have been Tom Kavli, at SINTEF, and Nils Christophersen, at the Department of Informatics. I thank them both for valuable guidance and assistance into the world of empirical modeling.

I will also thank Glenn Lines for intense and fruitful discussions, Irene Rødsten, Jon von Tetzchner Stephenson, and Svein Linge for careful reading of the manuscript and for correcting my English, Geir Horn and John W. Bothner for helpful hints, and Erik Weyer for explaining obscure mathematical details. A special thank goes to the library at SINTEF for providing all necessary literature.

Oslo, August 17, 1994.

Lars Aarhus

[a]Set in 12pt Times Roman using LaTeX. Drawings made with `idraw` and plots with MATLAB.

Contents

List of Figures

List of Tables

Abbreviations

ANN	Artificial Neural Network
ASMOD	Adaptive Spline Modeling of Observation Data
BP	Back Propagation
CART	Classification And Regression Trees
CNLS	Connectionist Normalized Linear Spline
CV	Cross Validation
FPE	Final Prediction Error
GCV	Generalized Cross Validation
HSOL	Hierarchical Self-Organized Learning
LSA	Local Search Algorithm
LWR	Locally Weighted Regression
MARS	Multivariate Adaptive Regression Splines
MLP	MultiLayer Perceptron network
MLR	Multiple Linear Regression
MSC	Multiplicative Signal Correction
MSE	Mean Square Error
MSECV	Mean Square Error of Cross Validation
MSEE	Mean Square Error of Estimation
MSEP	Mean Square Error of Prediction
NARMAX	Nonlinear AutoRegressive Moving Average with eXogenous inputs
NIPALS	Nonlinear Iterative PArtial Least Squares
NIR	Near InfraRed
NRMSEE	Normalized Root Mean Square Error of Estimation
NRMSEP	Normalized Root Mean Square Error of Prediction
PCA	Principal Component Analysis
PCR	Principal Component Regression
PLS	Partial Least Squares regression
PPR	Projection Pursuit Regression
RAN	Resource-Allocation Network
RBFN	Radial Basis Function Network
RMSEE	Root Mean Square Error of Estimation
RMSEP	Root Mean Square Error of Prediction

Notation index

A matrix is written with boldface, uppercase letters, a vector (either row or column) with boldface, lowercase letters and a scalar with plain letters. When referring to samples, superscripts are used in the symbols.

Symbol	Description	Reference
a, A, A_m	Index, latent variables, $a = 2, ..., A$ (for local model m)	Section 2.2.2
A_w	Principal components in weighting subspace	Section 4.2
$\mathbf{b}, b_p, b_{mp}, b_{m0}$	Regression coefficients (for local model m)	Section 2.2.1
C	Sample covariance	Appendix B.2
$\mathcal{D}$	Data set of input and output samples	Section 4.2
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$	Data set of training (test) samples	Section 2.1.3
$\mathcal{D}_v$	Cross validation data set of training samples	Section 2.1.4
$e^n, e^n_{(1)}, \bar{e}^n_{(K)}, e^n_{(K)}$	Sample error n	Section 4.2
f, f_m	Approximation function (for local model m)	Section 2.1 (3.4)
f_v	Cross validation approximation function	Section 2.1.4
$\hat{f}$	Underlying unknown function	Section 2.1
F	Independent parameters (degrees of freedom)	Section 2.1.4
g, g_a, h_a	Functions	Section 2.2.3
$\mathbf{I}$	Identity matrix	Section 3.3
i, j	General indices	
J	Validation criterion	Section 2.1.4
K	Nearest neighboring samples	Section 2.2.3 and 4.2
l, L	Index, candidate samples $l = 1, 2, ..., L$	Section 4.6
m, M	Index, local models $m = 1, 2, ..., M$	Section 3.1
n, N	Index, samples, $n = 1, 2, ..., N$	Section 2.1
$N_{\text{train}}, N_{\text{test}}, N_m$	Samples in training (test) set (for local model m)	Section 2.1.3 (4.2)
O	Original input space	Section 4.2

Symbol	Description	Reference
p, P	Index, input variables, $p = 1, 2, ..., P$	Section 1.1
$\mathbf{p}_a$	Loading column vector in PLS	Section 2.2.2
$\mathbf{P}$	Covariance matrix of input or latent variables	Section 3.3
q_a	Output loading	Section 2.2.2
$\bar{r}$	Average correlation	Appendix B.1
$r_m, r_{\max}$	Radius (for local model m)	Section 4.3
$\mathbf{R}$	Correlation coefficient matrix of input variables	Appendix B.1
s_m	Splitsample index	Section 4.2
S	Sample standard deviation	Appendix B.2
$\mathbf{t}$	Latent variable row vector	Section 2.2.2
$\mathbf{t}^n, \mathbf{t}^i$	Projected sample row vector n (i)	Section 4.2
$\mathbf{t}_a$	Principal component or score a (column vector)	Section 2.2.2
t_a	Latent variable a	Section 2.2.2
$\mathbf{T}, \mathbf{T}_{\text{test}}$	Projected input training (test) matrix	Section 2.2.2 (4.2)
v, V	Index, cross validation subsets $v = 1, 2, ..., V$	Section 2.1.4
$v_{ap}, \mathbf{v}_a$	Weights, weight column vector	Section 2.2.2
w_m	Weights (for local model m)	Section 3.3
$\mathbf{w}_a$	Loading weight column vector	Section 2.2.2
W	Weighting principal component input subspace	Section 4.2
$\mathbf{x}$	Input variable row vector	Section 2.1
$\mathbf{x}^n, \mathbf{x}^i$	Input sample row vector n, (i)	Section 2.1
$\mathbf{x}_p$	Input variable column vector p	Section 2.1
x_p	Input variable p	Section 2.1
$\mathbf{X}, \mathbf{X}_{\text{test}}, \mathbf{X}_m$	Input training (test) matrix (for local model m)	Section 2.1 (4.2)
y	Output variable	Section 2.1
y^n, y^i	Output sample n, (i)	Section 2.1
$\hat{y}$	Prediction of output variable	Section 2.1.4
$\mathbf{y}, \mathbf{y}_{\text{test}}, \mathbf{y}_m$	Output training (test) column vector (for local model m)	Section 2.1 (4.2)
Λ	Smoothing matrix	Section 3.3
λ, λ_m	Smoothing parameter (for local model m)	Section 3.3
μ, μ_m	Center of validity function (for local model m)	Section 3.3 and 4.2
ρ, ρ_m	Validity function (for local model m)	Section 3.3

1

Introduction

Empirical modelling can be defined as the problem of establishing a mathematical model or description of a *system* merely from a limited number of observations of different variables in the system. The model is usually an input/output model where some variables, named *input* variables, are used to predict the response of the remaining one(s), named *output* variable(s). The system is either static, i.e. is in a fixed condition, or dynamic, i.e. undergoes an evolution in time.

One example of its use is in the field of chemical processing industry, where complicated chemical processes with unknown relationships between process variables are investigated. Empirical modelling can then be used to gain insight in these relationships. Other fields where empirical modelling is applied today include:

- Chemometrics
- NIR spectroscopy
- Image processing
- Classification
- Control systems
- Geology
- Economy

The main reason for applying empirical modelling in these fields is because *analytical* models, which generally are the most desirable, are either highly inaccurate or very difficult to derive. Both is the case when one has very little a priori knowledge about the system. In these fields, an analytical model would also be very complex since the number of input variables is often very high. One is therefore left with the second-best alternative, empirical modelling, in order to interpret and understand the connections between the variables involved in the system. This is important since a model based on better understanding will often lead to a decrease in the error of prediction, and can in the end even reduce the total expenses of a company, if for instance, the system one is modeling is part of an industrial process.

1.1 Motivation

When doing empirical modeling there are many different techniques and algorithms available. One common way of classifying them is to separate them into global and local techniques. In a *global* modeling technique the idea is to find *one* function which describes the relationship between the variables. This function will be valid in the whole input domain. Examples of such techniques are Multiple Linear Regression (MLR) [Johnsen and Wichern 88], Principal Component Regression (PCR) [Martens and Næs 91], and Partial Least Squares Regression (PLS) [Wold *et al.* 84, Geladi and Kowalski 86, Lorber *et al.* 87, Höskuldsson 88].

The idea in a *local* modeling scheme is to find local functions which only describe the relationship between the variables in a local domain in the input space. These local models can then be interpolated, yielding a global description of the system. Local modeling schemes employ the principle of *divide-and-conquer.* This principle states that the solution to a complex problem can be solved by dividing it into smaller independent problems. These problems are then solved, and their solutions are combined yielding the solution to the original complex problem. Examples of local techniques are Locally Weighted Regression (LWR) [Næs *et al.* 90, Næs and Isaksson 92], Artificial Neural Networks (ANN) with Radial Basis Functions (RBF) [Moody and Darken 88, Stokbro *et al.* 90], the ASMOD algorithm [Kavli 92] and the LSA algorithm [Johansen and Foss 93]. All the techniques are treated in greater detail in chapter 2.

To illustrate the idea behind empirical modeling and also the difference between global and local modeling, a simple example is given. Consider the intuitively easiest problem to solve, namely the static univariate case, i.e. a single input variable, x, and a single output variable, y. All one is given is N corresponding observations of the two variables. Hence the problem can be formulated as finding the best relationship between x and y based on the N observations. It might be instructive to think of the observations as data points in a two dimensional space, see figure 1.1a.

If linearity is assumed in x, the most common solution is to fit a straight line through the set of data e.g. by the method of least squares. The result is a *linear* relationship between x and y, described by the *line of regression* ,i.e. the slope b and the y-intercept b_0. Thus, the global empirical model is $y = b_0 + bx$, see figure 1.1b.

However, if the set of data clearly shows a nonlinear behavior, as is the case in this example, a better approach might be to fit a *nonlinear* function to the set of data, giving a nonlinear global empirical model. An example of such is the quadratic model $y = b_0 + b_1x + b_2x^2$, where the parameters b_0, b_1, and b_2 are again found by the method of least squares. The resulting curve on our set of data is given in figure 1.1c.

Another interesting alternative when solving this nonlinear problem, is to divide the N observations into a number of different groups based on their value along the x-axis. By assuming linearity and performing linear regression within each group, local linear regression models are formed. This approach is an example of local empirical modeling. The result for our set of data can be seen in figure 1.1d. Here the number of groups is 3, and the local models are interpolated using a weight function to avoid piecewise linear functions.

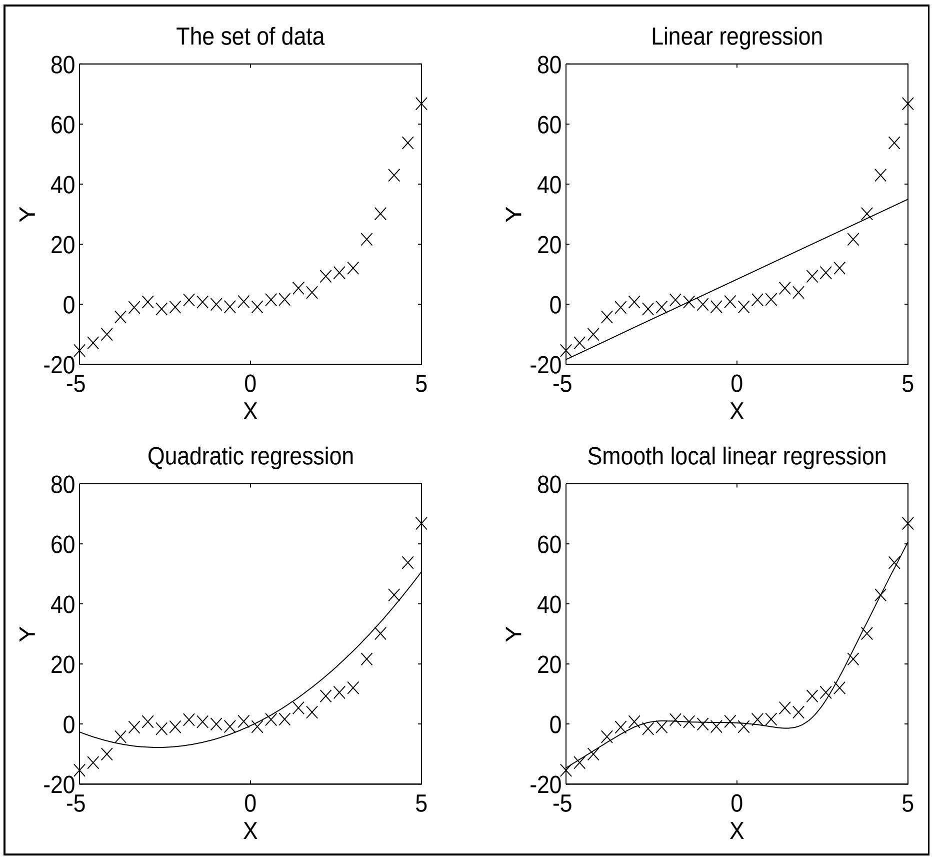

Figure 1.1: Simple empirical modeling with 26 observations. (a) The set of data. (b) Linear regression. (c) Quadratic regression. (d) Smooth local linear regression.

This example can easily be expanded to the multivariate case with P input variables, but still only a single output variable, i.e. a mapping from $\mathcal{R}^P$ to $\mathcal{R}$. Apparently this system is more complex than in the univariate case, but the same kind of thinking concerning nonlinearity/linearity and global/local models can still be applied. A further expansion to multiple number of output variables can be done, simply by modeling one y-variable at a time. The term *multivariate system* will from now on refer to a system with P input variables and one output variable.

1.2 Overview and scope of thesis

This thesis presents some new nonlinear empirical modeling algorithms, all based on local modeling. As seen from the example in section 1.1, the way of thinking that nonlinearity can be approximated by local linearity is appealing both because it is simple and because it is not very computationally demanding. The proposed algorithms are all iterative when it comes to finding the local models. They are constructed for a general framework, and are not directed towards any particular application or problem, although emphasis is on prediction. A nonlinear connection between input and output variables is always assumed. High dimensional ($P > 3$), noisy problems are of special interest, especially when the input variables are correlated.

The rest of the thesis is organized as follows:

- The next chapter takes a closer look at general problems when doing empirical modeling. In addition, some of the most important existing modeling techniques are presented.
- Chapter 3 covers problems that are specific to *local* modeling.
- In chapter 4, the proposed algorithms and the philosophy behind them are described and tested on artificial examples.
- The results obtained by applying the best of the proposed methods on four well-known data sets, are given in chapter 5.
- A further discussion on some of the algorithms and an evaluation of their performance takes place in chapter 6.
- The last chapter contains the main conclusions of the work in this thesis.

2

Background

This chapter provides a general background to empirical modeling, as seen from the multivariate approximation point of view [Poggio and Girosi 90]. First, the problem formulation is specified, and some important aspects which often cause problems in the modeling are presented. Empirical modeling as a two-step process is then described. The most important linear techniques, as well as a review of different nonlinear techniques are presented. Local modeling, which can be seen as a special class within nonlinear modeling, is introduced in greater detail in chapter 3.

2.1 Multivariate approximation

The general problem treated in this thesis can be formulated as a multivariate approximation problem. In our context this means finding the best possible nonlinear relationship between the vector of input variables, $\mathbf{x} = (x_1, ..., x_P) \in \mathcal{R}^P$, and the scalar output variable, $y \in \mathcal{R}$. The relationship will be of the form

$$y \approx \hat{y} = f(\mathbf{x}) \tag{2.1}$$

where f is a nonlinear approximation function, and $\hat{y}$ is the predicted value of y. This definition is motivated by the assumption that there exists an underlying function, $\hat{f}$, from which both $\mathbf{x}$ and y are generated. Unfortunately, $\hat{f}$ is unknown to us.

To help us find f, the only information that is available is N different observations, or samples, of $\mathbf{x}$ and y. In other words $\mathbf{x}^n, n = 1, 2, ..., N$ and $y^n, n = 1, 2, ..., N$. Usually N is greater than P, but there are situations e.g. in spectrometry where this is not always true.

The data can be arranged in two matrices, denoted $\mathbf{X}$ and $\mathbf{y}$. $\mathbf{X}$ is a $N \times P$ matrix having $\mathbf{x}^1$ to $\mathbf{x}^N$ as row vectors, whereas $\mathbf{y}$ is a $N \times 1$ matrix (column vector) consisting of y^1 to y^N. The column vectors of $\mathbf{X}$, i.e. sample values from one input variable, are denoted $\mathbf{x}_p, p = 1, 2, ..., P$.

Another way to formulate the general problem is by trying to visualize the situation geometrically. All the corresponding samples of $\mathbf{x}$ and y can then be thought of as N geometrical points spread out in a $P+1$ dimensional hyperspace, having orthogonal axes formed by the $P+1$ variables. The solution to the problem of identifying f is then the best P-dimensional *hyperplane* fitted to all the points, if f is to be globally linear, or more generally, the best *manifold*, if one is looking for a nonlinear model.

An important reason why only an approximate, and not an exact relationship can be found, is because of disturbances or *noise* in the samples. The presence of noise is responsible for a number of undesirable phenomena, such as overfitting, outliers, and the bias/variance problem. The purpose of any approximation function, f, is to filter away as much noise as possible, but at the same time to keep the underlying structure in the system.

2.1.1 Variables and samples

The variables x_1,...,x_P and y are modelled as *stochastic* variables corrupted by noise, because there is always an element of chance in the real world, where no system is completely deterministic. Hence the description of the variables includes statistical terms such as mean and variance. However, no assumptions are made about the underlying probability density functions from which the observations are generated.

Three important aspects of stochastic variables are expected to cause difficulties in the problem context of this thesis:

Internal correlation. The different input variables are often strongly internally correlated. This might cause problems when using algorithms like MLR, which assumes that $\mathbf{X}$ has full rank i.e. no or insignificant collinearity in $\mathbf{X}$. The result is unstable parameter values and basis for serious misinterpretations of the model, f [Dempster *et al.* 77]. One common way to overcome this problem is to project the samples in the P-dimensional hyperspace onto a lower dimensional hyperspace spanned by orthogonal, uncorrelated variables. PCR and PLS are two algorithms which use this concept of projection.

High dimensionality. When the number of input variables, P, is higher than at least 3, one talks about a *high dimensional* set of data, and a corresponding high dimensional hyperspace of samples. In such a hyperspace things do not behave as nicely as in a simple two or three dimensional space. One particular problem is that the higher the dimension, i.e. the larger P is, the more sparsely populated with observation samples, the hyperspace tends to be. In fact, if the dimension is increased by one, one needs an exponential growth in the number of samples, N, in order to ensure the same density, d. ($N = d^P$) Similarly, the number of parameters required to specify the model, f, will also increase exponentially with P. This is known as the *curse of dimensionality.* To avoid this curse, one could always assure oneself that one has a sufficiently large number of samples. However, this is an unrealistic approach since N is a number which is most likely to be fixed. Instead, the solution is either to reduce the dimensionality by projections (e.g. PCR and PLS), to decompose the input variables by expressing f as a sum of lower dimensional combinations of the input variables (e.g. MARS [Friedman 88] and ASMOD), or to put strong constraints on the model complexity.

Outliers. Since the variables are corrupted by noise, some of the samples could show some types of departure from the rest of the data. Such samples are called *outliers* or abnormal observations. The question is what to do with samples like these. Should they simply be removed from the sample collection, or, on the contrary, be regarded as the most important carriers of information? And on what basis should such a decision be made? There is no simple solution here, especially not when doing nonlinear modeling where the difference between what is noise and what is a nonlinear trend is much smaller. Therefore, removal of outliers in nonlinear modeling is more dangerous than in linear modeling, and should only be done with extreme care.

2.1.2 Properties of f

A solution f, to equation 2.1, should have the following generally desirable properties:

- First of all, f should give an as good as possible *prediction*, $\hat{y}$, of y when presented with a new input vector, $\mathbf{x}$. This is the main objective when developing prediction models.

- Secondly, f should be *parsimonious* with as few parameters and local models as possible. This is in accordance with the parsimony principle of data modeling [Seasholtz and Kowalski 93], which states:

  > If two models in some way adequately model a given set of data, the one described by a fewer number of parameters will have better predictive ability given new data.

  This principle is also known as Occam's razor.

- Lastly, f should be a *smooth* function. By smooth is meant throughout this thesis $\mathcal{C}^1$. A smooth f will have better generalization properties than discontinuous or piecewise smooth models [Poggio and Girosi 90], i.e. it will make better predictions of y. Another reason for requiring smoothness is that the underlying relationship, $\hat{f}$, which one is trying to model, is in fact often assumed to be smooth. Hence it appears only natural that f should be smooth as well.

2.1.3 Training (finding f)

The process of finding f is referred to as the learning or training step of multivariate approximation. This step consists of finding both f itself, and the set of parameters in f which provides the best possible approximation of $\hat{f}$ on the set of training samples. These training samples either equals the N samples previously defined or are a subset of these. The latter is the case if the N samples are divided into disjunct training and test sets. From now on, the number of training samples is in any case denoted N_{train}. Likewise, the set of training samples, known as the training set, is denoted $\mathcal{D}_{\text{train}}$. The test set, if present, is denoted $\mathcal{D}_{\text{test}}$, with the number of test samples denoted by N_{test}.

The main problems which immediately occur in the training step are essentially those of approximation theory and are listed below:

Model structure The determination of the model structure is the first and foremost task. With model structure is meant which function, f, to use in the approximation. Is a linear f sufficient, or must a nonlinear be used? What one usually does is to start with a simple linear model, and then try more complex models if the linear approach was not a good choice. This is known as the representation problem. Determining the number of layers and nodes in an ANN is an example of this problem.

Model algorithm and parameters Once the model structure is determined, the next problem is to decide which mathematical algorithm to use to find the optimal values of the parameters for a given f, and then finding these values. Usually, the choice of algorithm is guided by the choice of model structure, but sometimes there is no dependency between algorithm and structure. For instance, MLR, PCR and PLS are all algorithms that produce linear models in $\mathbf{x}$, just in different ways. Often, these linear models are not even similar, but they are still all linear. The parameter values are estimated by the algorithm. In this process the model structure must satisfy specific criteria which put constraints on the parameter values. Such criteria can be least squares fit, continuity, smoothness etc.

The choice of algorithm and model structure will very much depend on what kind of problem one is investigating, since no algorithm is universally the best.

Another aspect is the efficiency of the algorithm. There is only seldom use for an algorithm which might give very good models, but at a high computational cost, compared to another which computes f in a fraction of that time and with only slightly worse results in terms of prediction ability. Examples of the former are different ANN, which are still slow in comparison with e.g PLS.

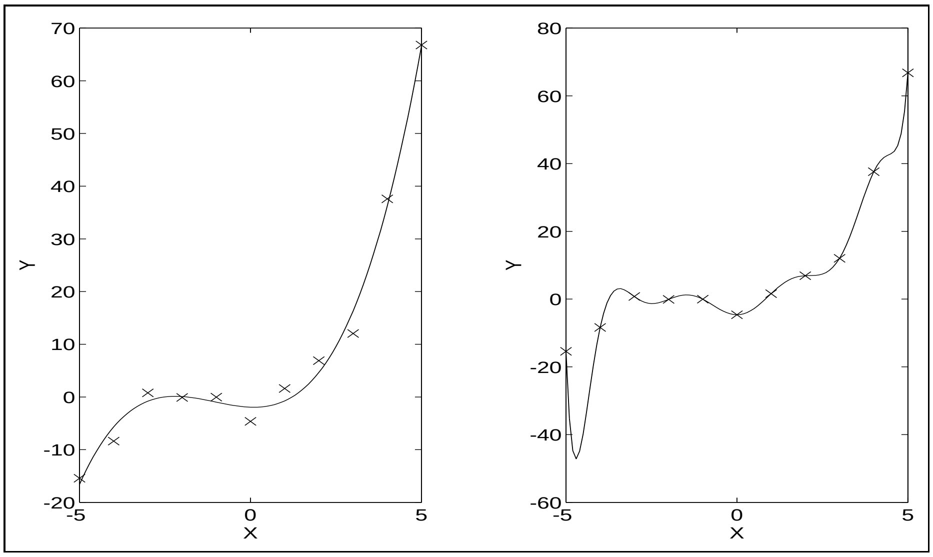

Figure 2.1: Overfitting. (a) A good fit to noisy data. (b) An overfitted model.

2.1.4 Testing (validating f)

Once the training step is over and a new model, f, is derived, the second important step in empirical modeling can start. This step is referred to as the validation or testing step, and consists of validating f against certain requirements. Whether or not f meets these requirements will decide whether f is a good model for our purpose or not. Usually one is interested in the predictive ability of f on new input samples, $\mathbf{x}$. Such a requirement can be specified in terms of a *validation criterion.*

One reason for validating f is to avoid *overfitting*, i.e. modeling of noise as well as underlying structure. The problem of overfitting happens when too much effort is put into fitting f to the training set. An illustration of an overfitted model and another which is not, on the same training set, is given in figure 2.1. The overfitted model in fact interpolates between the training samples because too many free parameters are allowed in f. The result of overfitting is a much *worse* prediction ability on new input samples. Since an overfitted model attempts to model both the noise and the system, overfitting is more likely to happen the more noise is present in the samples. For a system without noise, overfitting will generally not be a problem.

Since one wants to measure the generalization properties, the ideal validation criterion would be to minimize the expected mean square error (MSE) between the 'true' output, y, and the predicted output, $\hat{y}$ given by

$$J^*_{MSE} = E\left[(y - \hat{y})^2\right] \tag{2.2}$$

However, minimizing J^*_{MSE} can not be done analytically since the probability density functions for the variables are unknown. An *estimate* of J^*_{MSE} must therefore be minimized.

Several such estimators exist, the most used is the *empirical* mean square error defined by

$$J_{MSE} = \frac{1}{N} \sum_{n=1}^{N} (y^n - \hat{y}^n)^2 \tag{2.3}$$

A very tempting approach is to use the N_{train} samples in the training set in the computation of J_{MSE}. The estimator is then known as the mean square error of estimation (MSEE). Unfortunately, this estimator will give biased estimates of J^*_{MSE} because the training set, $\mathcal{D}_{\text{train}}$, is used both in the training and in the testing step. The estimates will simply be too 'good'.

The alternative is to compute J_{MSE} from an independent set of test samples. Usually $\mathcal{D}_{\text{test}}$ is another subset of the original N samples, with $N = N_{\text{train}} + N_{\text{test}}$ and $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ being disjunct sets.

This estimator, known as the mean square error of prediction (MSEP), will be unbiased, and is therefore one of the most used validation criteria. It is important, though, that the samples in $\mathcal{D}_{\text{test}}$ are representative of the system one attempts to model. This means they should be distributed in the hyperspace in the same way as the samples in $\mathcal{D}_{\text{train}}$, otherwise MSEP will not be a good measure of the prediction ability.

The main drawback using an independent test set is that these samples will no longer be available to us in the training step. This is not a problem if one has a large amount of data, but is not desirable in situations when data are sparse or costly to collect, which unfortunately is the case for many modeling problems. In such situations one would like to use all N samples in the training step. One solution is then to use all the training samples in the validation step once, but not all at the same time. This method is known as *V-fold cross-validation* [Stone 74]. In this approach the training set, $\mathcal{D}_{\text{train}}$, is randomly divided into V subsets of nearly equal size, denoted $\mathcal{D}_v, v = 1, 2, ...V$. Then, in addition to the original model f based on the whole training set, V other models denoted by $f_v, v = 1, 2, ..., V$, are found simultaneously. Each f_v is found using the $V - 1$ subsets $\mathcal{D}_1 + ... + \mathcal{D}_{v-1} + \mathcal{D}_{v+1} + ... + \mathcal{D}_V = \mathcal{D}_{\text{train}} - \mathcal{D}_v$ as the training set. The prediction ability of f_v is then tested on the remaining subset, $\mathcal{D}_v$, which will act as the independent test set. The V-fold cross-validation estimator [Breiman *et al.* 84] is given by

$$J_{MSECV} = \frac{1}{V} \sum_{v=1}^{V} \text{MSE}(f_v, \mathcal{D}_v) \tag{2.4}$$

where $\text{MSE}(f_v, \mathcal{D}_v)$ is the mean square error, defined by equation 2.3, of subset $\mathcal{D}_v$ using f_v as the model.

The main advantage using cross-validation is that it is parsimonious with data, since every sample in $\mathcal{D}_{\text{train}}$ is used as a test sample exactly once. There is no need for a separate test set anymore. However, it is important that V is large for J_{MSECV} to yield a good estimate. Thus, cross-validation is a computer intensive method, which is a disadvantage. Note that with $V = N$, the 'leave-one-out' cross-validation is obtained. This is also known as full cross-validation.

Two other often used estimators of J^*_{MSE}, which are also computed from the samples in $\mathcal{D}_{\text{train}}$ only, are the Final Prediction Error (FPE) criterion [Akaike 69] given by

$$J_{FPE} = \frac{1}{N}\sum_{n=1}^{N}(y^n - \hat{y}^n)^2 \Big/ \left(\frac{1-F/N}{1+F/N}\right) \tag{2.5}$$

and the Generalized Cross Validation (GCV) criterion [Craven and Wabha 79] given by

$$J_{GCV} = \frac{1}{N}\sum_{n=1}^{N}(y^n - \hat{y}^n)^2 / (1-F/N)^2 \tag{2.6}$$

where F is the effective number of independent parameters (degrees of freedom) in the model, f. Both these criteria are particularly useful in iterative algorithms since they penalize models with complex model structure and many parameters. The drawback is that a good estimate of the degrees of freedom, F, is difficult to compute since many of the parameters will often be more or less dependent. One way of doing it is suggested in [Friedman 88] and applied in his MARS algorithm.

Because the squared prediction error may be difficult to interpret, one often prefers to talk about the square root of the estimated MSE, which is named RMSE (root mean square error). The advantage is that this estimate is measured in the same unit as y itself.

2.2 Modeling techniques

In this section some of the most important existing approaches to empirical modeling are presented. Focus is on describing the training algorithm, and specifying the form of the model, f, it produces. In addition, since no technique always is the best, it is mentioned when the techniques work well and under what circumstances they fail.

All the algorithms presented below work best when the variables $\mathbf{x}_1$ to $\mathbf{x}_P$ and $\mathbf{y}$ are all normalized with respect to mean, variance etc. This can be done in many ways [Martens and Næs 91], but in this thesis it is assumed that the variables are *autoscaled* i.e. mean centered and scaled to variance one. The main reason for normalizing is to let each variable have an equal chance of contributing in the modeling.

This pretreatment obviously changes the matrices $\mathbf{X}$ and $\mathbf{y}$ defined earlier. An element, $\mathbf{X}_{np}$, in $\mathbf{X}$ is now equal to $(\mathbf{X}^{old}_{np} - \bar{x}(\mathbf{x}_p))/S(\mathbf{x}_p)$. However, to avoid too much notation the new autoscaled matrices will also be referred to by $\mathbf{X}$ and $\mathbf{y}$, and whether it is meant the unscaled or autoscaled versions will rather explicitly be stated. For the rest of this chapter $\mathbf{X}$ and $\mathbf{y}$ are autoscaled matrices.

2.2.1 Multiple Linear Regression

The classical linear approach to the problem formulated in section 2.1 is Multiple Linear Regression (MLR) [Johnsen and Wichern 88]. As the name indicates, this technique is ordinary linear regression of the output variable, y, on the set of P input variables, $\mathbf{x}$. The model, f, is then linear in $\mathbf{x}$ and of the form

$$f(\mathbf{x}) = \sum_{p=1}^{P} b_p x_p = \mathbf{x}\mathbf{b} \tag{2.7}$$

where $\mathbf{b} = (b_1, ..., b_p)^T$ are the regression coefficients given by the least squares solution $\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

The problem using MLR is that the input variables need to be linearly independent to give a unique solution. If they are not, the inverse matrix, $(\mathbf{X}^T\mathbf{X})^{-1}$, will be singular. This has already been discussed in section 2.1.1. As a rule of thumb, MLR should never be used if strongly correlated input variables are suspected, since the inverse matrix then might be close to singularity.

2.2.2 Linear projection techniques

To better cope with both internally correlated variables and high dimensional set of data, a class of projection techniques has been developed during the last decades. What they all have in common, is that they aim to model y by projections of the original set of input data onto a subspace spanned by a set of A orthogonal *latent variables*, where A is usually much less than P. These new variables are always computed as linear combinations of the original P input variables. The output variable, on the other hand, is either a linear or nonlinear combination of the latent variables in the subspace. This relationship between the output and latent variables is called the *inner relation*.

Thus, the whole idea behind projection techniques can be seen as reducing the dimensionality of the problem as much as possible, losing as little as possible of the essential information in the observations. Another useful feature is that the possibilities of graphical inspection of the set of data now have improved. In all the projection techniques, different two and three dimensional *plots* of e.g. the latent variables are important tools for interpreting the observations.

Two of the most popular multivariate projections techniques are Principal Component Regression (PCR) and Partial Least Squares Regression (PLSR or just PLS). They both use a *linear* inner relation. The model, f, is then of the form

$$f(\mathbf{x}) = \sum_{a=1}^{A} b_a \left(\sum_{p=1}^{P} v_{ap} x_p \right) = \sum_{a=1}^{A} b_a t_a = \mathbf{t}\mathbf{b} \tag{2.8}$$

where the v_{ap}'s are the weights of factor a and the b_a's are the regression coefficients of y on the vector of latent variables, $\mathbf{t} = (t_1, ..., t_A) \in \mathcal{R}^A$.

PLS and PCR differ in *how* the parameters v_{ap} are found, as will be explained in more detail below. Note first that in neither PLS nor PCR the parameters are estimated by fitting equation 2.8 as well as possible, as this would only lead to ordinary MLR. Instead, quite different algorithms are applied.

Principal Component Regression

In PCR, principal component analysis (PCA) [Wold *et al.* 87] is used to select the latent variables. The first principal component, $\mathbf{t}_1$, is defined as the projection of $\mathbf{X}$ onto the normalized eigenvector, $\mathbf{w}_1$, corresponding to the largest eigenvalue of $\mathbf{X}^T\mathbf{X}$. In other words, $\mathbf{t}_1 = \mathbf{X}\mathbf{w}_1$, where $\mathbf{w}_1$, the loading weight vector, can be seen as the direction spanning most of the variability in $\mathbf{X}$.

The other principal components, $\mathbf{t}_p, p = 2, ..., P$, are defined in the same way, as successive projections of $\mathbf{X}$ onto the other normalized eigenvectors, $\mathbf{w}_p, p = 2, ..., P$, under the constraint that the principal components are all orthogonal. These eigenvectors correspond to the respective eigenvalues of $\mathbf{X}^T\mathbf{X}$ in descending order, and they will be orthogonal as well.

Instead of computing all the eigenvalues simultaneously by e.g. singular value decomposition of $\mathbf{X}$, they are often computed successively in descending order, because one is only interested in a few of them. This can be done using e.g. the NIPALS algorithm [Wold 66]:

1. Initialization: $\mathbf{X}_0 = \mathbf{X}$ (assumed to be scaled and centered)
2. for factor $a = 1, 2, ...A$ compute loading vector $\mathbf{w}_a$ and score vector $\mathbf{t}_a$ as:
   (a) Initial estimate: $\mathbf{t}_a =$<column in $\mathbf{X}_{a-1}$ with highest sum of squares>
   (b) repeat until <eigenvalue estimate convergence>
      i. Improve estimate: $\mathbf{w}_a = \left(\mathbf{t}_a^T\mathbf{t}_a\right)^{-1} \mathbf{X}_{a-1}^T\mathbf{t}_a$
      ii. Scaling: $\mathbf{w}_a = \mathbf{w}_a \left(\mathbf{w}_a^T\mathbf{w}_a\right)^{1/2}$
      iii. Improve estimate: $\mathbf{t}_a = \mathbf{X}_{a-1}\mathbf{w}_a$
      iv. Eigenvalue estimate: $\mathbf{t}_a^T\mathbf{t}_a$
   (c) Subtract the effect of this factor: $\mathbf{X}_a = \mathbf{X}_{a-1} - \mathbf{t}_a\mathbf{w}_a^T$

Since there are P eigenvalues of $\mathbf{X}^T\mathbf{X}$ there will be P principal components. However, only the first A components are interesting since they will contain all the significant variability in $\mathbf{X}$. They are ordered in the matrix $\mathbf{T} = (\mathbf{t}_1, ..., \mathbf{t}_A)$, which can be thought of as the input matrix $\mathbf{X}$ in compressed form. In PCR, the latent variables are nothing more than these A principal components, where A is usually selected by cross-validation or test set validation.

In the final step in PCR, the output variable, y, is regressed on the latent variables using ordinary MLR, giving the regression coefficients $b_a, a = 1, 2, ..., A$ in the general equation 2.8. For each a, the weights, $v_{ap}, p = 1, 2, ..., P$, in the same equation are equal to the elements in the loading weight vector, $\mathbf{w}_a$.

PCR can be characterized as an *unsupervised* method, since the latent variables are not found using information about the output variable, y. Instead, PCR is a *variance maximizing* method, because only those latent variables which contribute the most to the variability in $\mathbf{X}$ are considered.

Partial Least Squares Regression

Contrary to PCR, PLS [Wold *et al.* 84, Geladi and Kowalski 86, Lorber *et al.* 87, Höskuldsson 88] is a *supervised* method, where the influence of y is incorporated when the latent variables are found. PLS can also be seen as a *covariance maximizing* method, since it maximizes the covariance between $\mathbf{X}_{a-1}\mathbf{w}_a$ and $\mathbf{y}$ under the same constraint as in PCR, $\mathbf{w}_a^T\mathbf{w}_a = 1$. In other words, the latent component, $\mathbf{t}_a$, is found by projecting $\mathbf{X}_{a-1}$ onto the direction $\mathbf{w}_a$, which now is a cross between the direction with highest correlation between the input variables and output variable (MLR approach) and the direction with largest variation in the input variables (PCR approach) [Stone and Brooks 90].

As with the NIPALS algorithm in PCA, the latent variables are computed successively in the PLS algorithm. Since PLS will be part of the new algorithms proposed in this thesis, the orthogonalized form of the algorithm is given below.

1. Initialization: $\mathbf{X}_0 = \mathbf{X}$ and $\mathbf{y}_0 = \mathbf{y}$ (both assumed to be scaled and centered)
2. for factor $a = 1, 2, ...A_{\max}$ compute loadings $\mathbf{w}_a$, $\mathbf{p}_a$ and q_a and score $\mathbf{t}_a$ as:
   (a) Loading weights: $\mathbf{w}_a = \mathbf{X}_{a-1}^T\mathbf{y}_{a-1}$
   (b) Scaling: $\mathbf{w}_a = \mathbf{w}_a\left(\mathbf{w}_a^T\mathbf{w}_a\right)^{1/2}$
   (c) Scores: $\mathbf{t}_a = \mathbf{X}_{a-1}\mathbf{w}_a$
   (d) Loadings: $\mathbf{p}_a = \left(\mathbf{t}_a^T\mathbf{t}_a\right)^{-1}\mathbf{X}_{a-1}^T\mathbf{t}_a$
   (e) Output loading: $q_a = \left(\mathbf{t}_a^T\mathbf{t}_a\right)^{-1}\mathbf{y}_{a-1}^T\mathbf{t}_a$
   (f) Subtract the effect of this factor: $\mathbf{X}_a = \mathbf{X}_{a-1} - \mathbf{t}_a\mathbf{p}_a^T$ and $\mathbf{y}_a = \mathbf{y}_{a-1} - \mathbf{t}_a q_a$
3. Determine A, $1 \leq A \leq A_{\max}$, the number of factors to retain.

In this algorithm the score vectors, $\mathbf{t}_a$, and loading weight vectors, $\mathbf{w}_a$, are orthogonal, whereas the extra loading vectors, $\mathbf{p}_a$, are generally not. A nonorthogonal form of the PLS algorithm exists as well, where no extra loadings are needed, but resulting in nonorthogonal latent variables or scores. Note that for neither of the two forms there is a straight-forward relationship, for each a, between the weights $v_{ap}, p = 1, 2, ..., P$ in the general equation 2.8 and the elements in the loading weight vector, $\mathbf{w}_a$. Instead the relationship is complex, and also includes the elements in the other loading vector, $\mathbf{p}_a$. For the orthogonalized algorithm it is given iteratively by $\mathbf{v}_1 = \mathbf{w}_1$ and $\mathbf{v}_a = (\mathbf{I} - \sum_{i=1}^{a-1}\mathbf{v}_i\mathbf{p}_i^T)\mathbf{w}_a, a = 2, ..., A$, where $\mathbf{v}_a = (v_{a1}, ..., v_{aP})^T$. The regression coefficients, b_a, are equal to the output loadings, $q_a, \forall a$.

The PLS algorithm shares two more common features with the PCR approach. The number of latent variables are selected by cross-validation or test set validation, and in the limit $A = P$, the PLS (and PCR) solution equals the MLR solution, i.e. equation 2.8 reduces to 2.7.

A modified version of the orthogonalized algorithm, known as the PLS2 algorithm, has been developed for the case when there is more than one output variable, but this is beyond the scope of this thesis. For a more comprehensive description of multivariate projection techniques see the textbook by [Martens and Næs 91].

2.2.3 Nonlinear techniques

MLR, PCR and PLS were presented in detail above, partly because they must be considered the three most frequently used *linear* multivariate modeling techniques today and partly because they will all be part of the new local modeling algorithms proposed in this thesis. When doing *nonlinear* multivariate modeling, though, the number of different techniques is much higher, and no technique can be said to be well established. A selection of some of the most common techniques are now described. A few of these are of special importance, as they are used as reference techniques, when evaluating the performance of the best of the proposed algorithms in chapter 5.

Nonlinear projection techniques

In [Næs *et al.* 93] different types of nonlinear projection techniques are discussed. This presentation is motivated from that article.

One simple way of introducing nonlinearity in the model is by either transforming the input variables, or augmenting the input matrix, $\mathbf{X}$, with higher order and cross terms of the original input variables, and then using this new $\mathbf{X}$ in the PCR or PLS algorithm. However, such augmentation is only useful if P originally was very small because of the exponential growth in P when all cross terms are included.

Another way is by keeping the PLS algorithm for dimensionality reduction purposes, but replacing the linear inner relation in equation 2.8 with a nonlinear one, yielding the following general form of f,

$$f(\mathbf{x}) = \sum_{a=1}^{A} g_a \left(\sum_{p=1}^{P} v_{ap} x_p \right) = \sum_{a=1}^{A} g_a(t_a) \tag{2.9}$$

where the g_a's are smooth nonlinear functions. This is first suggested in [Wold *et al.* 89], using quadratic polynomials without cross terms. The idea is further developed in [Wold 92] where the smooth functions are approximated by splines. The result is that fewer latent variables are sufficient to describe the variability in $\mathbf{X}$, but at the expense of a much slower algorithm.

Quadratic PLS regression is also suggested in [Höskuldsson 92b]. However, this approach differs from the one in [Wold *et al.* 89], both because cross-terms are allowed in the polynomials, and more importantly because the selection of the quadratic PLS factors is based on the so-called H-principle of modeling data [Höskuldsson 92a].

Another technique, which is essentially based on the same model as in equation 2.9, is Projection Pursuit Regression (PPR) [Friedman and Stuetzle 81]. In this technique, the g_a's are totally general functions except that they are smooth. As in the PLS algorithm, one factor or latent variable, with weights $v_{ap}, p = 1, 2, ..., P$ and function g_a, is computed at a time. The effect of this factor is subtracted from $\mathbf{y}_{a-1}$, and the same procedure is applied to the residuals, $\mathbf{y}_a$. However, contrary to the PLS algorithm, there are no orthogonal restrictions in the estimation of the v_{ap}'s in the iterative procedure. A drawback with PPR is that the predictor f can not be written in closed form because the g_a's are only smooth fits to the samples in $\mathcal{D}_{\text{train}}$, usually determined using moving averages. Prediction of y for new samples must therefore be made by interpolations between the training samples. For further discussion of projection pursuit in general, and PPR in particular see [Huber 85].

A technique called nonlinear PLS is proposed in [Frank 90]. This approach is also essentially based on the same model as in equation 2.9, with the g_a's being determined by a smoothing procedure, as in PPR. However, the $\mathbf{v}_a$'s are estimated under exactly the same strong restriction of covariance maximization as in PLS, which makes this technique a kind of hybrid of PPR and PLS. The same drawback as in PPR, regarding prediction of y for new samples, is present in this approach.

Locally Weighted Regression

In [Næs *et al.* 90] a technique is suggested which is a generalization of the PCR algorithm, replacing the last MLR step with a locally weighted multiple linear regression (LWR), thereby the name. To be more specific, first the original input hyperspace is projected onto a lower dimensional hyperspace spanned by the latent variables $t_a, a = 1, 2, ...A$, using a standard PCA of $\mathbf{X}$. A new input sample, $\mathbf{x}^i \in \mathcal{R}^P$, will then correspond to a sample, $\mathbf{t}^i \in \mathcal{R}^A$, in the latent hyperspace. The K nearest neighboring *projected* samples among the N_{train} samples in the *training* set are then selected, based on their Mahalanobis distance (see appendix B.3) to $\mathbf{t}^i$. These samples are given a weight between 0 and 1, using a cubic weight function, again depending on their relative distance to $\mathbf{t}^i$. At last, a MLR is performed based on the K weighted samples and the corresponding K output samples. The result is a local prediction model, f^i, of essentially the same form as equation 2.8, which is now used to predict y^i. A new such local model must be computed for *each* single prediction, since another input sample, $\mathbf{x}^j$, will lead to different weighted neighboring samples and thus a different local regression model. The optimal number of neighboring samples, K, and principal components, A, can both be determined using cross-validation or test set validation.

LWR is a locally linear, but globally nonlinear projection technique. The drawback is again that the predictor, f, can not be written in closed form because the prediction of y for new samples must include the presence of the training samples in $\mathcal{D}_{\text{train}}$.

In [Næs and Isaksson 92] some modifications to LWR are suggested, including a new distance measure and a uniform weight function.

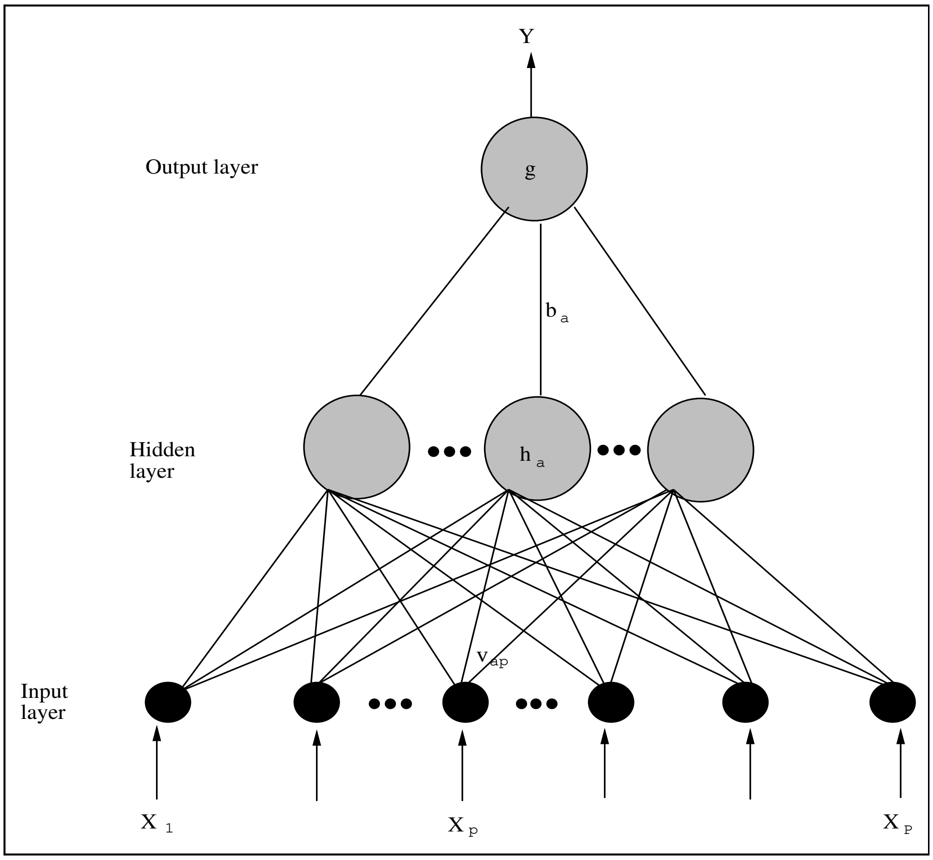

Figure 2.2: Artificial neural network with one hidden layer and one output node.

Artificial Neural Networks

The field of Artificial Neural Networks (ANN) covers many different network algorithms, and its use has exploded in the last decade. For a good survey and other references see the textbook by [Hertz *et al.* 91]. ANN has shown a lot of potential in modeling arbitrary nonlinear relationships. The terminology of ANN is somewhat different from other techniques. These new terms will be introduced by pointing to the illustration in figure 2.2.

Two of the most common types of networks are Multilayered Perceptron Networks (MLP) [McClelland *et al.* 86] and Radial Basis Function Networks (RBFN) [Moody and Darken 88, Stokbro *et al.* 90]. Both are feed-forward networks, where the information (i.e. samples) from the input layer is passed through intermediate variables (hidden layers) to the output layer. These intermediate variables can be thought of as projections or transformations of the original input variables. In the figure there is only one hidden layer. Each layer consists of a number of *nodes*. This number equals P in the input layer and is one in the output layer. In the hidden layer(s) there are no restrictions on the number of nodes, A. Each node, except those in the input layer, is usually connected with all the nodes in the previous layer. These connection lines will all have different *weights*, denoted v_{ap} and b_a in the figure. In each node, the weighted information from the previous layer is transformed by *transfer functions*, denoted h_a and g, before being passed to the next layer. The transformation is very different in MLP and RBFN. The output from the network will have the general form

$$f(\mathbf{x}) = g\left(\sum_{a=1}^{A} b_a h_a \left(\sum_{p=1}^{P} v_{ap} x_p\right)\right) \tag{2.10}$$

A bias term can also be added to each node in the hidden and output layer before the transformation, but this is not illustrated in the figure nor in equation 2.10.

In MLP the transfer functions are typically sigmoid shaped e.g. $h_a(z) = \tanh(z)$ or $h_a(z) = 1/(1 + \exp(-z))$. The most common learning algorithm is error back propagation (BP) [Rumelhart *et al.* 86], which is a gradient descent algorithm finding the optimal weights by usually minimizing the sum of squared estimation sample errors, or a variation thereon.

In RBFN there is always only one hidden layer, all the v_{ap}'s are equal to 1 and g is the identity transformation. Thus, the input to the transfer functions h_a is no longer a weighted *sum*. The model f is then reduced to the form

$$f(\mathbf{x}) = \sum_{a=1}^{A} b_a h_a(r) \tag{2.11}$$

where the h_a's are scalar radial basis functions centered around the extra parameter vectors μ_a and $r = \|\mathbf{x} - \mu_a\|_{\mathrm{M}}$. Examples of basis functions are the logarithmic function $h_a(r) = \log(r^2 + c^2)$, where c is a constant, and the Gaussian function $h_a(r) = \exp(-\frac{1}{2}r^2)$. Again, a gradient descent algorithm is usually applied to iteratively estimate the network parameters. A good overview of RBFN is given in [Carlin 91].

Observe that the different learning algorithms in ANN are just ways of finding the optimal parameters from the training samples, which are presented to the network in random order. Generally, this is a slow procedure compared to other nonlinear techniques.

A comparison between MLP and RBFN has shown that MLP is slower, requires more layers, but less samples and hidden nodes than RBFN to obtain the same level of accuracy [Moody and Darken 89]. In addition, RBFN does not work too well when data are high dimensional because of the necessity of a distance measure in the radial basis functions, h_a.

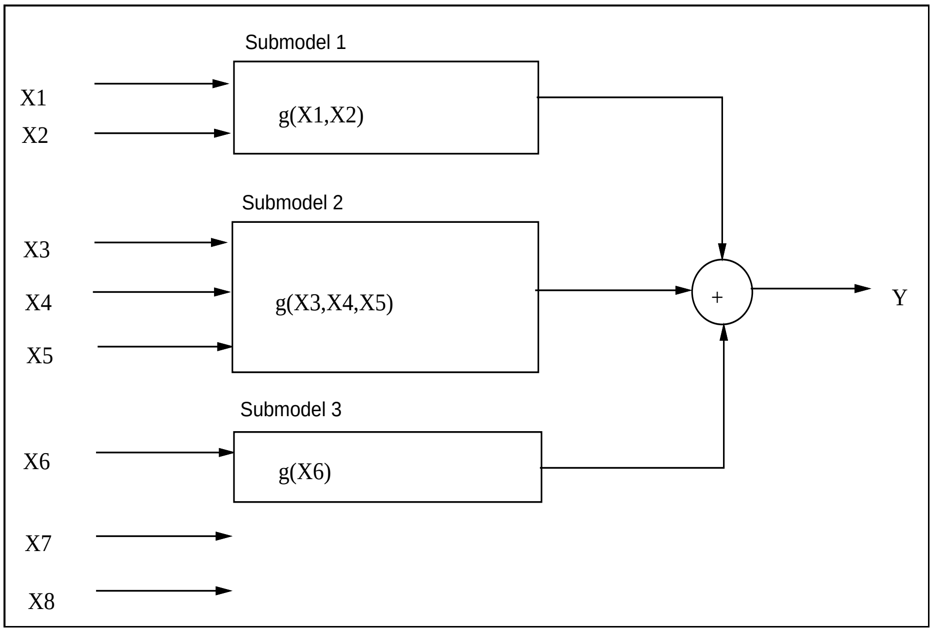

Figure 2.3: The MARS and ASMOD model structure.

Adaptive spline techniques

Multivariate Adaptive Regression Splines (MARS) [Friedman 88] and Adaptive Spline Modeling of Observation Data (ASMOD) [Kavli 92] are two techniques that represent f by the decomposition exemplified in figure 2.3. The general form of the model is given by

$$f(\mathbf{x}) = \sum g_i(x_i) + \sum g_{ij}(x_i, x_j) + \sum g_{ijk}(x_i, x_j, x_k) + ... \tag{2.12}$$

In both algorithms, a subset of the possible submodels, g, are selected during the training process. Both apply *splines* in the function representation of the submodels, although MARS employs natural splines as opposed to B-splines which are used in ASMOD. Generally, splines have great abilities of approximating multivariate functions by joining polynomial segments (basis functions) to form continuous and smooth functions. For more comprehensive presentations of splines see [Farin 88, Wahba 90].

MARS is a two-step algorithm. The first step is a forward partition step decomposing the input space into a number of overlapping submodels. The second is a backward pruning step deleting those submodels which contribute the least in the fit based on the GCV criterion (see equation 2.6).

In ASMOD both of these steps are combined in one iterative model refinement procedure. In each step in the iteration either a new one-dimensional submodel is added, two submodels are replaced by one higher dimensional submodel, or a new knot is inserted in the B-spline of any of the submodels, depending on which of the three ways reduced the estimation error the most. The refinement is terminated when the prediction error is at a minimum.

This way of decomposing will include only the submodels for input variables that are necessary in the prediction of y. With strongly correlated variables only limited improvement of the predictions can be expected when more than a few of the variables are added to the model. Thus, the adaptive spline techniques aim at keeping the number of submodels to a minimum. At the same time, they can be seen as modeling high dimensional data by a sum of lower dimensional submodels, as the dimensionality of the submodels are kept as low as possible.

3

Local modeling

An interesting approach to nonlinear empirical modeling is *local* modeling. In this chapter the modeling philosophy behind this approach is presented and discussed. The presentation serves as an introduction to the proposed local modeling algorithms in chapter 4. The model, f, is then of the general form

$$f(\mathbf{x}) = \sum_{m=1}^{M} w_m(\mathbf{x}) f_m(\mathbf{x}) \tag{3.1}$$

where f_m is the local model, w_m the corresponding weight function, and M the total number of local models.

Local modeling is characterized by the decomposition of the input hyperspace into smaller local hyperspaces with equal dimension. A simple local model is found in each of these hyperspaces. Such a model will only describe local variations, since it will only be based on training samples *within* the local hyperspace. All these local models are then interpolated by the use of local weight functions, yielding the total model f. This model should have better predictive ability than a simple global model, otherwise there is no need for a local approach.

The major problems that are specific to local modeling are:

- How to divide the input hyperspace?
- How to decide the optimal number of local models, M?
- How to interpolate between the local models?
- How to represent the local models, using which algorithm?

These problems will be addressed in this chapter. A more statistical discussion of multivariate calibration when data are split into subsets is found in [Næs 91]. Two approaches to local modeling, which are related to the work of this thesis and have proved to be inspiring, are an algorithm using fuzzy clustering by [Næs and Isaksson 91] and the local search algorithm (LSA) in [Johansen and Foss 93]. Details about them will be presented in the subsequent sections in this chapter as examples of different ways of dealing with the problems mentioned above.

3.1 Division of input space

The most fundamental problem in local modeling is *how* to divide the input hyperspace in a parsimonious way such that the number of local models is kept at a minimum, but is still sufficient to adequately model the system, in the sense described in section 2.1.2. This problem is closely linked to that of how *many* local models the hyperspace should be divided into. As one has very little a priori knowledge about the system, the exact number and precise position of these models are not known in advance.

One possibility is to construct a *grid* in the hyperspace and then include local models around those grid points, where there are enough samples. However, this static approach is both time consuming and impractical in high dimensional spaces. With a uniform (equal spacing) and homogeneous distribution of grid points in every dimension, the result is an exponential growth in the number of grid points and local models (see section 2.1.1). Thus, even with relatively few grid points in each dimension, the number of local models will be very large. Another drawback is that all the local models will be valid in equal sized local hyperspaces. Often some of them can easily be replaced by a larger one, without reducing the overall effect.

A more dynamic approach is to apply a *clustering* algorithm. Such an algorithm seeks to cluster the samples together in M disjunct groups, by minimizing the total spatial (Euclidian or Mahalanobis) distance between all the samples within a group. The number of groups, M, does not always have to be known in advance, although it is often required. Many different algorithms are available, both hierarchical, nonhierarchical and a hybrid of those. For a good survey see [Gnanadesikan *et al.* 89].

One drawback using traditional clustering algorithms is that they only consider closeness in space when samples are assigned to different groups. This might be desirable in classification problems, but since the purpose of this thesis is prediction, and not classification, that aspect should also be reflected in the decomposition algorithm.

One way of doing this is suggested in [Næs and Isaksson 91]. There, a division of the input space is proposed based on a *fuzzy* clustering algorithm [Bezdec *et al.* 81], with M fixed. However, the distance measure is now a weighted combination of the Mahalanobis distance (from traditional clustering) and the squared residuals from local fitting of the samples. After the convergence of the clustering algorithm, each sample is allocated to the group for which it has the largest so-called membership value. This fuzzy clustering algorithm is part of an approach to local modeling which is based on many of the same principles as in LWR. First, the input hyperspace is projected using a standard PCA. Then, after the clustering algorithm is applied to all the projected training samples, a separate linear regression is performed within each group, resulting in M locally linear PCR models. However, these models are not interpolated, so f in equation 3.1 will not be smooth. Instead, new samples are simply allocated to the closest class by an ordinary Mahalanobis distance method in traditional discriminant analysis. The optimal number of local models is found by cross-validation or test set validation.

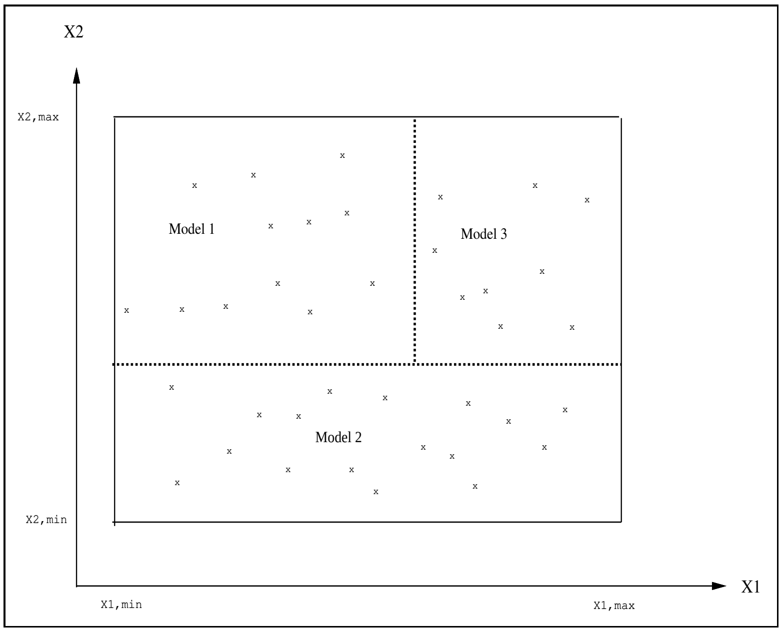

Figure 3.1: Splitting of the input space into rectangular boxes in the LSA algorithm.

Another approach is an *iterative* decomposition into smaller and smaller hyperspaces. A prime example is applied in the LSA algorithm [Johansen and Foss 93], which divides the input space into hyperrectangles (see figure 3.1 for an illustration). In each iteration one of the hyperrectangles is split into two smaller ones. Which hyperrectangle should be split and how is decided by testing several alternatives and choosing the one that gives the largest decrease in a validation criterion. The LSA algorithm also involves local linear models and smooth interpolation between the local models by the use of Gaussian functions.

The approach was originally developed for NARMAX models [Johansen and Foss 92a, Johansen and Foss 92b], but has been extended to general nonlinear dynamic and static models. The algorithm involves no projection of the input variables as a result of being developed in a system identification context. It is therefore best suited for lower dimensional problems, and all the present experience is on that kind of problems. However, as suggested in [Johansen and Foss 93], it can easily be expanded to high dimensional problems by first carrying out a principal component projection as is done in [Næs and Isaksson 91].

Other examples of iterative decompositions can be found in the MARS [Friedman 88] and CART [Breiman *et al.* 84] algorithms.

The clustering algorithms and the two approaches outlined above are all using so-called ‘hard’ splits of data. This means that each sample only belongs to one local hyperspace and thus only contributes to one local model. Such a split is known to increase the variance [Jordan and Jacobs 94]. The alternative is ‘soft’ splits of data, which allow samples to lie simultaneously in multiple hyperspaces.

3.2 Validation and termination of modeling algorithm

No matter *how* one divides the input space, there will always be need for validating an actual splitting because one wants to determine the optimal one. Optimal, in the sense that this splitting gives the largest improvement in prediction ability, compared to other splittings of the same input space and with different numbers of local models, M.

If M is fixed in the decomposition algorithm, the common approach is either cross-validation or test set validation.

However, in an iterative decomposition algorithm the validation must be done after *each* iteration since each step will produce a new splitting of the data and increase the number of local models by one. Another important aspect in an iterative algorithm is to determine *when* to end the refinement of the model, f.

These two tasks can be combined using either test set validation, cross-validation, or other types of validation (see section 2.1.4) and stopping the iteration when one of the estimators has reached a minimum value. An example is the application of the FPE and GCV criterion in the LSA algorithm [Johansen and Foss 93].

An alternative is to have a separate validation criterion and a stop criterion, which is based on the evolution of the estimation error (MSEE). One such stop criterion is to end the refinement when the MSEE is levelling out, i.e. the difference between the MSEE in the last and second-to-last step is lower than a predefined value. However, this is only recommended when the danger of overfitting is reduced due to relatively little noise in the data and when N_{train} is large.

Another approach is to stop iterating when the MSEE becomes smaller than a prespecified limit. This limit should be set slightly higher than the anticipated noise level, in order to avoid modeling random noise. This approach is only advisable when a good estimate of the noise level is available.

Both these somewhat ad hoc approaches are motivated by figure 3.2 which shows a typical behavior of the prediction error (MSEP) and MSEE, as the complexity of f, i.e. number of local models, increases. The figure also illustrates the effect of overfitting when random noise is modelled.

The very best approach when validating and terminating an iterative algorithm is to have *three*, all representative and independent, data sets:

- A training set, used to compute the parameters in f.
- A stop set, used to determine when to stop refining f in the training step.
- A test set, used to validate the final prediction ability of f.

If the data sets are representative this will be a completely unbiased estimate. Unfortunately, this approach is usually not feasible since it requires too much data.

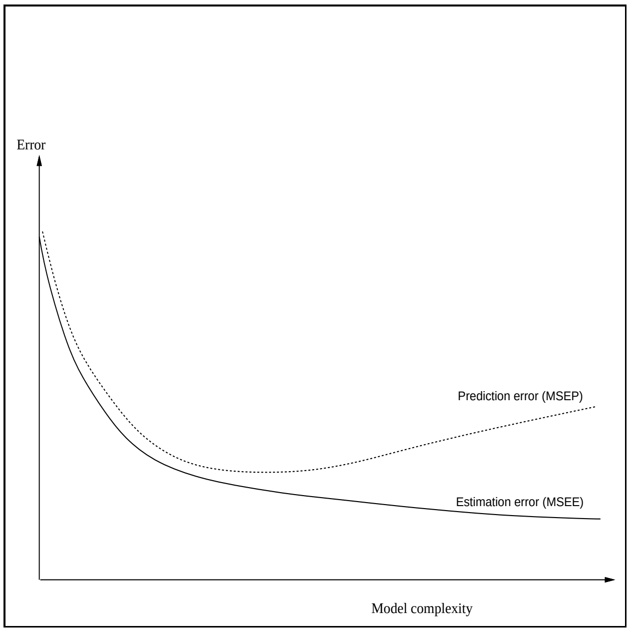

Figure 3.2: General evolution of the estimation error (MSEE), and the prediction error (MSEP), as a function of increasing model complexity.

3.3 Weighting and interpolation

All the local models will have a limited range of validity. One therefore needs to determine how to interpolate them to yield a complete global model, f. Having no interpolation at all will lead to a rough or even discontinuous f, which is undesirable.

One way is to assign a normalized and smooth weight or interpolation function to each local model. Such a function is often of the form

$$w_m(\mathbf{x}) = \frac{\rho_m(\mathbf{x})}{\sum_{j=1}^{M} \rho_j(\mathbf{x})} \tag{3.2}$$

where ρ_m is a scalar local validity function [Johansen and Foss 92a], which should indicate the validity or relevance of the local model as a function of the position of $\mathbf{x}$ in input space. Furthermore, ρ_m should be nonnegative, smooth, and have the property of being close to 0 if $\mathbf{x}$ is far from the center of a local model. The use of smooth validity functions, along with smooth local models, f_m, ensures that f will be *smooth* as well.

Ideally, the sum of the validity functions at any position, $\mathbf{x}$, in the input space should be equal to unity. This can be achieved in practice by normalizing the ρ_m's. One is then ensured that the total weight given to $\mathbf{x}$ from all the local models is always unity, because $\sum_{m=1}^{M} w_m = \sum_{m=1}^{M} \rho_m / \sum_{j=1}^{M} \rho_j = 1, \forall \mathbf{x}$. However, there is also a danger using this way of weighting, as extrapolation to regions in input space *outside* the operating regime of the system is now very easy. This is not always advantageous.

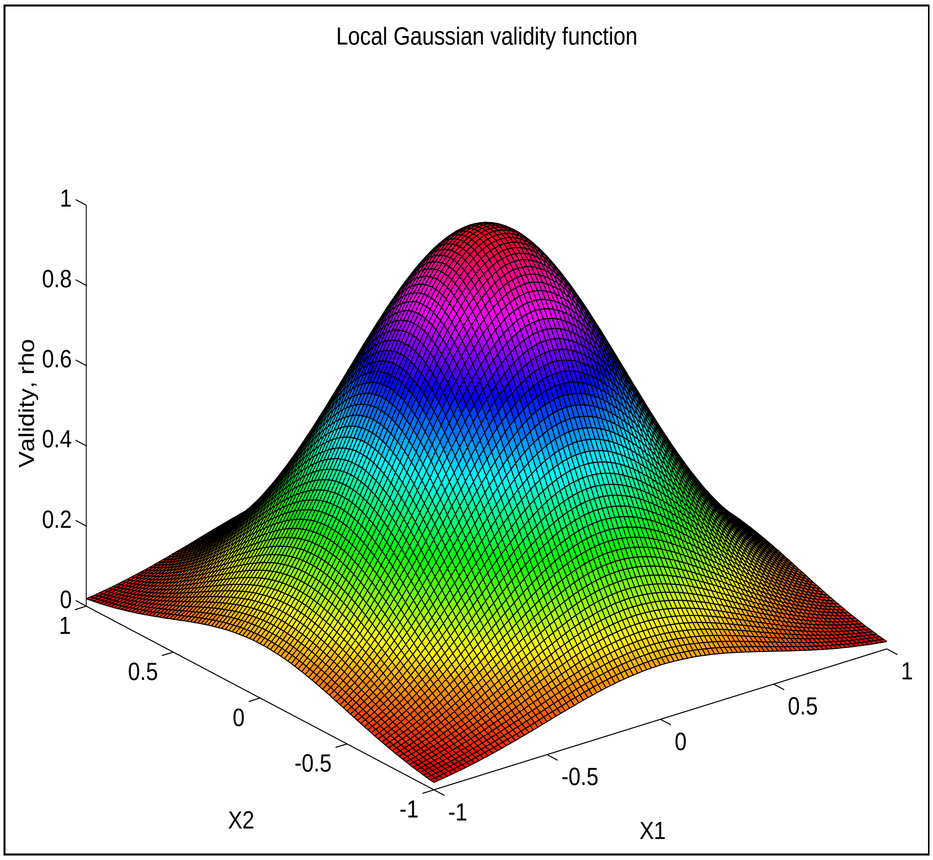

Figure 3.3: A two dimensional unnormalized Gaussian function.

Since the validity function is *centered* around a local model, one needs to define the center, μ_m. Usually, μ_m, is defined to be either the mean vector of all the training samples in class m (center of 'mass'), or the center of the local hyperspace itself if the space has properly defined boundaries (geographical center). One example of the latter is the hyperrectangular region in the LSA algorithm, where μ_m is the center of the box.

Probably the most used validity function is the unnormalized *Gaussian* function. The general form of this multivariate function is

$$\rho = \exp(-\frac{1}{2}(\mathbf{x}-\mu)\Lambda^{-1}(\mathbf{x}-\mu)^T) \tag{3.3}$$

where μ is the center of the local validity function and Λ is a *smoothing* matrix which will define the overlap between the different local validity functions. In one dimension, Λ can be thought of as the squared standard deviation or the width of the Gaussian function. A two dimensional example is given in figure 3.3.

There are many ways of choosing the smoothing matrix Λ:

1. Let $\Lambda = \lambda^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix and λ is a smoothing parameter. This parameter will be the *same* for every validity function $\rho_m, m = 1, 2, ...M$. The result is one single validity function, having *equal spherical* contour lines in the two dimensional case.

2. Let $\Lambda = \lambda^2\mathbf{P}$, where $\mathbf{P}$ is the covariance matrix of $\mathbf{X}$, the matrix of input training samples. Again, λ will be the *same* for every validity function $\rho_m, m = 1, 2, ..., M$, and the result is still one single validity function, but it will no longer have spherical contour lines. Instead it will have *equal elliptical* lines, and along the main axes if $\mathbf{P}$ is a diagonal matrix.

3. Let $\Lambda = \lambda_m^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. λ_m is an individual smoothing parameter, different for each validity function $\rho_m, m = 1, 2, ..., M$. The result is separate validity functions and *different spherical* contour lines. The problem is how to choose the λ_m's and the relationship between them.

4. Let $\Lambda = \lambda_m^2\mathbf{P}$, where $\mathbf{P}$ is the covariance matrix of $\mathbf{X}$. Again, λ_m is individual for each validity function $\rho_m, m = 1, 2, ..., M$, resulting in *different elliptical* contour lines, and along the main axes if $\mathbf{P}$ is a diagonal matrix. The problem is again to find a good way of choosing the different λ_m's.

5. Let $\Lambda = \lambda^2\mathbf{P}_m$, where $\mathbf{P}_m$ is the covariance matrix of the input training samples belonging to local model m. The result is not only elliptical contour lines but also individually *orientated* validity functions $\rho_m, m = 1, 2, ..., M$. In the other four approaches, the orientation is the same for all the validity functions, but here it is guided by the distribution of the local samples.

Examples of different types of contour lines are given in figure 3.4. Note that approach 1 is equal to 2, and 3 to 4, if $\mathbf{P} = \mathbf{I}$, i.e. $\mathbf{X}$ is an autoscaled matrix of uncorrelated variables. Approach 4 is also known from RBFN, where it is called *input specific standard deviation*.

An important question is how far into the domain of model m, should the surrounding models exert influence. This question is very much related to the choice of λ_m. Intuitively, when there are 'many' local models, there should be little overlap between them. On the other hand, with 'few' local models, a larger relative overlap is more appropriate. A tiny overlap is linked with small values of λ_m, whereas larger values will give more overlap between the local models and a more smooth f. A large value of λ_m also reduces the variance of f, but at the expense of a more biased model [Johansen and Foss 93]. The point is to find values that balances these phenomena.

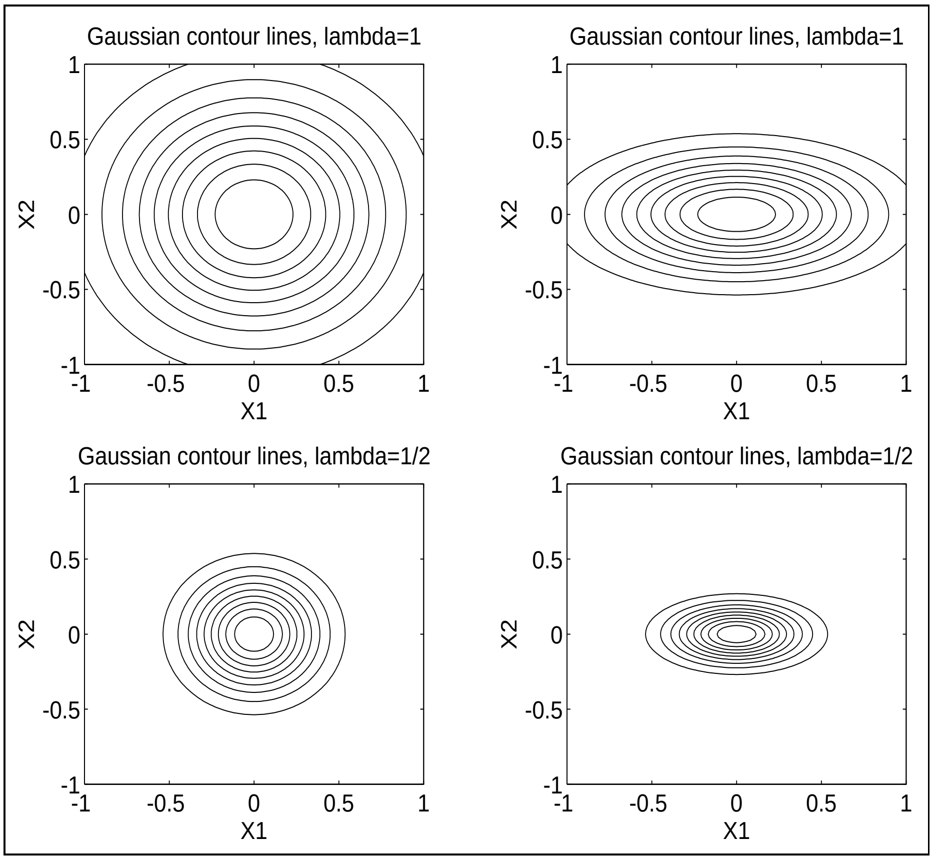

Figure 3.4: Gaussian contour lines for different choices of Λ. (a) Identity matrix, $\lambda = 1$. (b) Diagonal matrix, $\lambda = 1$. (c) Identity matrix, $\lambda = \frac{1}{2}$. (d) Diagonal matrix, $\lambda = \frac{1}{2}$.

Another aspect is that since distance metrics are involved, all weighting and interpolating should take place in a projected low dimensional space and not in the original, possibly high dimensional hyperspace. If PCA is utilized as the projection technique, reducing $\mathbf{X}$ to the latent matrix $\mathbf{T}$, the covariance matrix of $\mathbf{T}$ will have the favorable property of being diagonal in approaches 2 and 4. The validity functions (and weight functions) will then be functions of the latent variables, i.e. $\rho_m = \rho_m(\mathbf{t})$ (and $w_m = w_m(\mathbf{t})$), and not of the original input variables.

Typical other choices of validity function are linear, quadratic or cubic splines and other well-known kernel-functions [Hastie and Tibshirani 90]. A more odd choice is to use indicator functions. This is equivalent to giving no weight to all but one of the local models, which will have unit weight. An example is the local modeling approach of [Næs and Isaksson 91]. As already mentioned, the result is no longer a smooth f.

3.4 Local type of model and optimization

One of the advantages with local modeling is that the structure of the local models, f_m, does not need to be very complex. Usually simple *linear* models are sufficient. The model, f, is still a good nonlinear predictor. However, the complexity is not in the local models themselves, but in the interpolation between them.

In this thesis only local models with linear model structure in the input variables $\mathbf{x}$ are used. The next problem is, as it was in global modeling, which mathematical algorithm to use for estimating the local parameter values. Since high dimensional input samples are assumed, focus will be on projection techniques such as PCA. Two techniques are then evident. Either MLR of y on the latent input variables, $\mathbf{t}$, or PLS modeling of y on the original input variables, $\mathbf{x}$.

The first approach is used in the algorithm proposed in [Næs and Isaksson 91], and will produce local PCR models. Each local MLR takes place in the same latent input space as the weighting does. In the second approach one applies the original input variables in each local PLS. The number of latent variables will then possibly be different for each local model. However, all *weighting* is still done in the same latent space, as found by the PCA algorithm.

An important question is whether the parameters in the local models should be optimized locally or globally. In a local optimization, each local model is optimized separately and then weighted, yielding the global model, f. A global optimization will optimize all the local parameters simultaneously, and should provide a better approximation of y on the training samples. On the other hand, when f is presented with new samples the performance can be worse as local optimized models are often more representative of the local behavior of the system than globally optimized ones [Murray-Smith 94].

Using MLR as the local modeling technique, both local and global optimization can be done with a standard weighted least squares algorithm. However, if PLS is applied, a global optimization would be very time consuming since the global optimal combination of different numbers of local latent variables, from different local PLS algorithms, must be obtained by e.g. cross-validation. This is a formidable task even for a small number of local models. In that case only local optimization is feasible. What is still a problem, though, is that a PLS algorithm with individual weighting of the input samples, which would have been desirable, does not exist to the best of my knowledge.

Although the focus in this thesis is on local linear models, that does not completely rule out selecting *nonlinear* local models and a nonlinear algorithm. But, usually very little is gained in prediction ability with such an approach. At least when compared to the much increased computational cost, which is unavoidable once the step from linear to nonlinear local models is taken.

4

Proposed algorithms

In this chapter three new algorithms for local modeling are proposed. The first, a local search algorithm, is the main algorithm of this thesis. Four different versions of this algorithm are presented in detail, all of which were implemented and tested. The other two, a global search algorithm and an extended local search algorithm, were not implemented and are just briefly described. All the algorithms give rise to models of the general form given in equation 3.1.

Before starting to develop the algorithms, a few important choices regarding the structure of the algorithm and solution, f, have to be made. These choices are taken on the basis of the discussion in chapter 3.

- The algorithms are based on an *iterative* splitting of the input space into local regions, implying a gradual refinement of f.

- The local regions have a *flexible* shape, and are not restricted to e.g. hyperrectangular boxes. With more complex region boundaries it is anticipated that a smaller number of local models is needed, but at the expense of more parameters for describing the regions.

- 'Hard' splits of data are used, with the local regions being *disjunct*. Each training sample will then contribute to one and only one local model.

- The interpolation between the local models is done with *smooth*, normalized weight functions and Gaussian validity functions. The input set of data is typically *projected* onto a lower dimensional subspace based on its first few principal components. Local weighting is then carried out in this subspace.

- *Linear* functions are used in the local models, with PLS as the local modeling technique.

The total model, f, then has the specific form found by substituting equation 3.2 (normalized weight function) and 2.8 (PLS model) into equation 3.1 yielding

$$f(\mathbf{x}) = \sum_{m=1}^{M} \frac{\rho_m(\mathbf{t})}{\sum_{j=1}^{M} \rho_j(\mathbf{t})} \left(\sum_{a=1}^{A_m} b_{ma} \left(\sum_{p=1}^{P} v_{map} x_p \right) + b_{m0} \right) \tag{4.1}$$

where A_m is the number of latent variables in local PLS model m and b_{m0} is a constant term.

As the latent variables are only linear combinations of the input variables, equation 4.1 can be reduced to

$$f(\mathbf{x}) = \sum_{m=1}^{M} \frac{\rho_m(\mathbf{t})}{\sum_{j=1}^{M} \rho_j(\mathbf{t})} \left(\sum_{p=1}^{P} b_{mp} x_p + b_{m0} \right) = \sum_{m=1}^{M} w_m(\mathbf{t}) f_m(\mathbf{x}) \tag{4.2}$$

without loss of generality. This is the form of f used in the discussion.

It can be shown mathematically that f can approximate any so-called *measurable* function arbitrary well. The details are omitted here, but the reasoning is that any measurable function can be approximated arbitrary well by a piecewise constant function. Thus, in equation 4.2, setting $b_{mp} = 0, \forall m, p$ gives piecewise constant functions, and anything that can be approximated by piecewise constant functions can also be approximated by f, and anything that can be approximated by f can also be approximated by piecewise constant functions. The only condition is that the validity function ρ_m must be allowed to be the *indicator* function, θ_R, as well as e.g. the Gaussian. The complete proof is given in e.g. [Folland 84].

$$\theta_R(\mathbf{t}) = \begin{cases} 1 & \mathbf{t} \in R \\ 0 & \mathbf{t} \notin R \end{cases} \tag{4.3}$$

So, theoretically the model structure in equation 4.2 should be able to handle any type of empirical modeling problem, at least when the number of samples, N, and the number of local models, M, go towards ∞. Practically though, N and M are of course bounded, which limits the prediction ability, as does the presence of noise in the observations.

1. Preprocessing of the data set, $\mathcal{D}$.
2. Initialization.
   - Define validity function ρ_1.
   - Compute initial global linear model, $f^{(1)} = f_1$.
   - The number of local models, $M = 1$.
3. while <consistent decrease in validation criterion J> do
   - Find the local region where $f^{(M)}$ is modeling the worst.
   - Allocate training samples to the $M + 1$ different local regions.
   - Define new validity functions ρ_m.
   - Compute new local linear models, f_m, in these regions.
   - Interpolate, $f^{(M+1)} = \sum_m w_m f_m$.
   - Validate $f^{(M+1)}$ using validation criterion J.
   - Increment, $M = M + 1$.
4. Determine the optimal model, f.

Figure 4.1: General **Algorithm 1**.

4.1 General Algorithm 1

The approach can be described as an iterative structure identification algorithm, based on error analysis of samples. The general **Algorithm 1** consists of the steps given in figure 4.1.

A total of four different versions of this algorithm are proposed, but only two proved to work well. The difference between them is in the repeated step (3), which constitutes the heart of the algorithm. The other steps (1, 2, 4) are essentially the same all the time.

All the four steps are now explained in detail. To make the presentation as easy to grasp as possible, this is first done by describing the complete **Algorithm 1a**, which is the most important, in the next section. The other three algorithms, **1b**, **1c**, and **1d** are then explained in section 4.3.

4.2 Algorithm 1a

Preprocessing (step 1)

This step first includes removal of possible outliers from the data set, $\mathcal{D}$. Since this algorithm is based on error analysis of samples, abnormal observations could strongly influence the results of the algorithm.

Then, $\mathcal{D}$ is divided into separate training ($\mathcal{D}_{\text{train}}$) and test sets ($\mathcal{D}_{\text{test}}$), both equally well distributed and representative of the system that is to be modelled.

The last step in the preprocessing is to select the first few (2–4) principal components of the input training matrix as a subspace used for weighting between the local models. The components are found by PCA analysis and the number of components are denoted by A_w.

After these initial operations, there are six different matrices. The training samples in $\mathcal{D}_{\text{train}}$ are organized in the input matrix $\mathbf{X}$ and the output matrix $\mathbf{y}$, both of which are assumed to be scaled and centered. The test samples in $\mathcal{D}_{\text{test}}$ are gathered in the input test matrix $\mathbf{X}_{\text{test}}$ and the output test matrix $\mathbf{y}_{\text{test}}$. The variables in these two matrices are assumed to be scaled and centered with the same factors as the variables in the training matrices. In addition, the projected input training samples are assembled in the matrix $\mathbf{T}$. The last matrix is $\mathbf{T}_{\text{test}}$, consisting of the input test samples projected onto the weighting subspace. The original input space formed by the input training samples are denoted by O, and the projected input subspace by W. There is always a one-to-one correspondence between a sample $\mathbf{x}^i$ in O and a sample $\mathbf{t}^i$ in W.

Initial model (step 2)

An initial local linear model, f_1, based on all the samples in the training set, is computed by the PLS algorithm. Since there is only one local model this is also the total model, $f^{(1)}$. A local model validity function ρ_1 is defined, having the center in origo of W, i.e. $\mu_1 = \mathbf{0}$. Origo is the natural choice since $\mathbf{T}$ has zero column mean.

Finding local region (step 3)

The whole idea behind the approach is to insert a new local model where the current model, $f^{(M)}$, is predicting worst. In other words, identify the local region where the expected deviation $E[|y - \hat{y}|]$ is largest. This region in input space will be represented by one of the samples in the training set $\mathcal{D}_{\text{train}}$, named the *splitsample* and denoted $\mathbf{x}^{s_M}$, where s_M is the splitsample index. The new local model is computed around that sample.

Finding this splitsample is the purpose of this step. Different ways of doing this are proposed. But, first a few definitions that are essential for understanding the principles. All the definitions below are to be associated with *one* input training sample, $\mathbf{x}^n$.

i. Single error, $e^n_{(1)}$**.** Defined as the absolute error $|f^{(M)}(\mathbf{x}^n) - y^n|$, where $f^{(M)}$ is the total model after M iterations.

ii. Mean error, $\bar{e}^n_{(K)}$**.** Defined as the absolute mean error $|\sum_{k=1}^{K}(f^{(M)}(\mathbf{x}^k) - y^k)/K|$ of the K nearest neighboring samples to $\mathbf{x}^n$, using the Mahalanobis distance with covariance matrix of $\mathbf{T}$ in projected input subspace. $\mathbf{x}^n$ is included among the K samples.

iii. Median error, $e^n_{(K)}$**.** Defined as the median error of the single errors in the K nearest neighboring samples ($\mathbf{x}^n$ included) using the Mahalanobis distance.

iv. Consistent. A sample $\mathbf{x}^n$ is said to have a *consistent* error if the signs of the single errors of the K nearest neighboring samples all are either plus or minus.

The general term *sample error*, denoted by e^n, is from now on used when referring to one of the error measures of type **i.**, **ii.**, or **iii.**.

Of the three types, the single error is the one most sensitive to outliers, since only information about *the* sample, $\mathbf{x}^n$, is considered. The two other types use information from surrounding samples as well, when trying to describe the error around $\mathbf{x}^n$, and are therefore less sensitive to noisy samples. In particular, the median error will be unbiased of any outliers.

The last definition will put a further requirement on a possible splitsample, by demanding that the closest surrounding samples should all have the same error behavior. If a sample has a consistent error, it is an indication that $f^{(M)}$ really is not predicting well around that sample. The term is a heuristic to be used in combination with one of the three types of error measure, in order to further avoid selecting an outlier.

The strategy for determining the splitsample is now described. In section 4.6 an alternative strategy is proposed. That, however, turned out to be worse than this one.

Strategy 1

*Select the splitsample as the sample of all the N_{train} samples having the largest sample error of type **i.**,**ii.** or **iii.**.*

This strategy performs in each iteration a global search for the splitsample among all the training samples, computing a sample error for each of the samples.

One restriction is that a training sample can not be picked twice. The splitsample is therefore always selected as the sample with the largest error, not previously selected. In other words, the splitsample is picked among $N_{\text{train}} + 1 - M$ training samples and not N_{train} as stated in strategy 1.

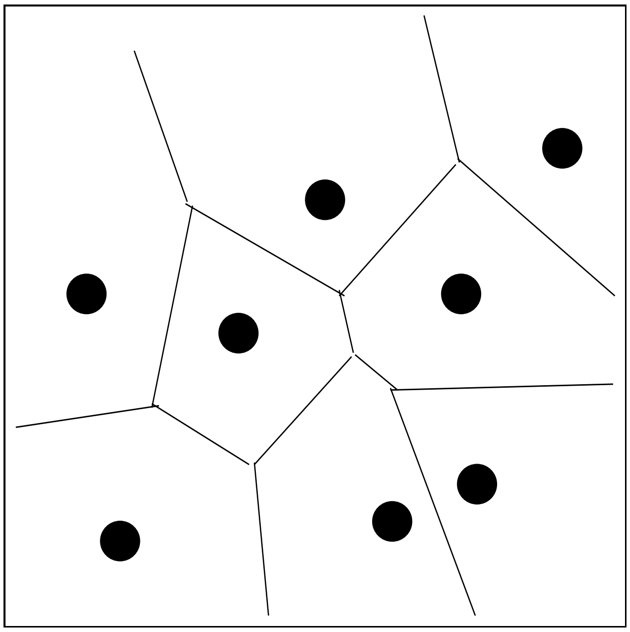

Figure 4.2: Local regions in input subspace, with boundaries and centers.

Allocation (step 3)

Once a new splitsample is found, a new validity function ρ_{M+1} is defined having the center in the projection of this splitsample, i.e. $\mu_{M+1} = \mathbf{t}^{s_M}$. There will then be a total of $M+1$ different validity functions with centers in $M+1$ geometrical points (M splitsamples plus origo) in W. These points define $M+1$ classes. The goal is to allocate all the samples in D_{train} to one, and only one, of these classes. This is done by allocating a sample to the class for which the validity function has the largest value, or expressed mathematically

$$\text{Class } m = \{\mathbf{x}^n, y^n | \arg \max_{j=1,2,\ldots,M+1} (\rho_j(\mathbf{t}^n)) = m\} \tag{4.4}$$

This classification divides up the projected input space into polyhedral regions as shown in figure 4.2. The division is known as a Voronoi (or Dirichlet) tessellation, and is often used in data compression [Hertz *et al.* 91].

Local model computation (step 3)

Since global optimization of the parameters is difficult when PLS is the local modeling technique (see section 3.4), local optimization is applied. Each local PLS model is found using the *original* input samples and not the projected ones.

The local training samples allocated to class m are gathered in a local input matrix $\mathbf{X}_m$ and output matrix $\mathbf{y}_m$, with N_m being the number of local samples. Before a local model f_m is computed the matrices $\mathbf{X}_m$ and $\mathbf{y}_m$ must be centered as required by the PLS algorithm. The local constant term b_{m0} is then equal to $(\bar{y}_m - \bar{\mathbf{x}}_m\mathbf{b}_m) = (\bar{y}(\mathbf{y}_m) - \sum_p b_{mp}\bar{x}(\mathbf{x}_{mp}))$.

The number of latent variables in each PLS model, A_m, is determined by local cross validation. All the local models $f_1, f_2, ..., f_{M+1}$ must be computed in each iteration since one never knows whether the allocation of samples to some regions is the same as in the previous iteration. The weights $w_1(\mathbf{t}^n), w_2(\mathbf{t}^n), ..., w_{M+1}(\mathbf{t}^n)$ (see equation 3.2) belonging to sample $\mathbf{x}^n$ are not used in the optimization process. Note that the validity functions, $\rho_1, \rho_2, ..., \rho_{M+1}$ are applied both in the allocation of samples and in the weighting of the local models.

Validation (step 3)

To investigate the total model $f^{(M+1)}$ in each iteration, both the root mean square error of estimation (RMSEE) and the root mean square error of prediction (RMSEP) are computed. The iteration is stopped when no further improvement is expected i.e. when the RMSEE might still be decreasing, but the RMSEP is *consistently* increasing. With consistently increasing is meant that the RMSEP increases in two preceeding steps. Admittedly this is a somewhat ad hoc stop criterion. The reason is to avoid an early termination because of local minima.

Optimal model (step 4)

The optimal model, f, is defined to be the one where the corresponding RMSEP is at a minimum. This RMSEP value is also used as the validation value of f. As noted earlier, doing this is a bit questionable, since the test set $\mathcal{D}_{\text{test}}$ will no longer be unbiased. The best would have been to have a third set of data, independent from the other two, and validate f on that set.

The entire **Algorithm 1a** is summarized in figure 4.3. The algorithm was implemented with all the three types of error measure and the consistent definition. A few assumptions regarding the local PLS modeling had to be made. They can be found, along with some further details about the implementation, in appendix A.

1. Preprocessing:
   (a) Removal of possible outliers from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$.
   (b) Training matrices: $\mathbf{X}$ and $\mathbf{y}$ (both assumed to be scaled and centered).
   (c) Test matrices: $\mathbf{X}_{\text{test}}$ and $\mathbf{y}_{\text{test}}$ (both scaled and centered with same factors as for the training matrices).
   (d) PCA analysis: Determine A_w from $\mathbf{X}$.
   (e) Projected input matrices: $\mathbf{T}$ and $\mathbf{T}_{\text{test}}$.
2. Initialization:
   (a) Define validity function: $\rho_1 = \exp(-\frac{1}{2}(\mathbf{t}-\mu_1)\Lambda^{-1}(\mathbf{t}-\mu_1)^T)$ where $\mu_1 = \mathbf{0}$, $\Lambda = \lambda^2\mathbf{P} =$, and $\mathbf{P}$ is the covariance matrix of $\mathbf{T}$.
   (b) Compute global PLS model $f^{(1)} = f_1$ from $\mathbf{X}$ and $\mathbf{y}$.
   (c) Number of local models, $M = 1$.
3. while <consistent decrease in validation criterion J> do
   (a) Find splitsample: $\mu_{M+1} = \mathbf{t}^{s_M}$ where $s_M = \arg\max_{n=1,\ldots,N_{\text{train}}(\neq s_1,\ldots,s_{M-1})}(e^n)$ and $e^n = e^n_{(1)}$ or $\bar{e}^n_{(K)}$ or $e^n_{(K)}$ (and $\mathbf{t}^{s_M}$ is consistent).
   (b) Define validity function: $\rho_{M+1} = \exp(-\frac{1}{2}(\mathbf{t}-\mu_{M+1})\Lambda^{-1}(\mathbf{t}-\mu_{M+1})^T)$.
   (c) for samples $n = 1, 2, \ldots, N_{\text{train}}$ do
      - Allocate sample $\mathbf{x}^n$ to $\mathbf{X}_m$ and y^n to $\mathbf{y}_m$ using equation 4.4.
   (d) Local training matrices: $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{M+1}$ and $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{M+1}$ (all assumed to be centered).
   (e) for region $m = 1, 2, \ldots, M+1$ do
      - Compute local PLS model f_m from $\mathbf{X}_m$ and $\mathbf{y}_m$.
   (f) Interpolate: $f^{(M+1)} = \sum_m w_m f_m$.
   (g) Validate $f^{(M+1)}$ using validation criterion $J = \text{RMSEP}(f^{(M+1)}(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}})$.
   (h) Increment: $M = M + 1$.
4. Optimal model: $f = \arg\min_{f^*=f^{(1)},\ldots,f^{(M-1)}}(\text{RMSEP}(f^*(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}}))$.

Figure 4.3: **Algorithm 1a** (Fixed validity functions).

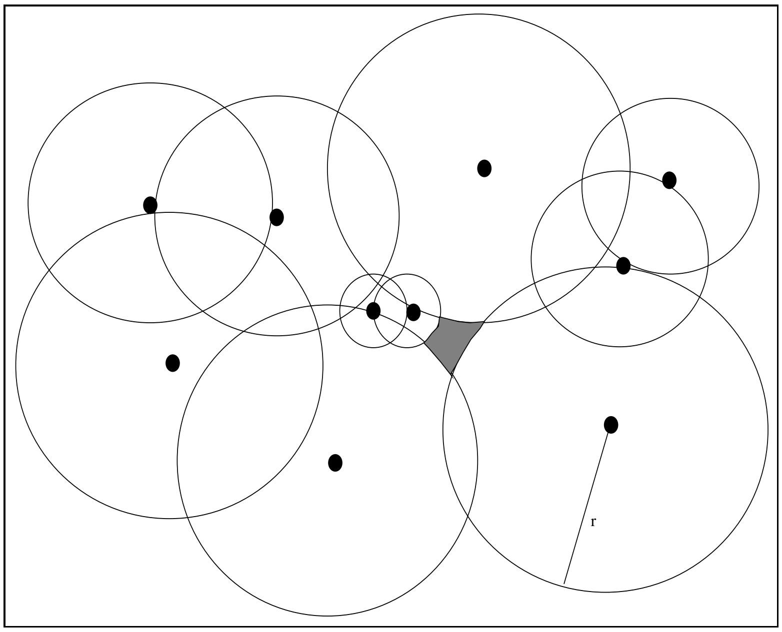

Figure 4.4: The concept of radius in local modeling (Identity matrix).

4.3 Algorithm 1b, 1c, and 1d

In the approach so far, the local validity functions have been fixed (see point 2 in section 3.3) and 'hard' splits of the training samples have been used. This is rather rigid. A more flexible approach is perhaps 'soft' splits and different sized validity functions, e.g. depending on the distance between or density of such functions.

In this section three modified versions of **Algorithm 1a** are presented. The changes in the algorithm only involve the allocation and local computation steps. How to find the splitsample and how to validate are not altered.

The distances between the centers of local validity functions are essential in all the three modifications. To each local model an extra parameter named the *radius*, and denoted r_m, is specified. This radius is defined to be the smallest Mahalanobis distance, with covariance matrix of $\mathbf{T}$, between the local splitsample, μ_m, and all the other splitsamples (origo included), or expressed mathematically

$$r_m = \min_{j=1,\ldots,m-1,m+1,\ldots M}(\|\mu_j - \mu_m\|_{\mathrm{M}}) \tag{4.5}$$

As previously noted, the local splitsample and the center of the local validity function are the same. An illustration of this approach is given in figure 4.4.

Both allocation of training samples and definition of validity functions can be based on this new term.

Allocate to one class, m, all the training samples within distance r_m from the center of that class, μ_m, and use those samples in the computation of the local model. One training sample can then be allocated to several different classes, or it does not have to be classified at all, which happens when the sample is in a region not 'covered' by the radii of any of the local classes (e.g. the shaded region in figure 4.4). 'Soft' splits are the result. The only requirement is that a minimum number of samples have to be allocated to each class, in order to avoid abortion of the algorithm because of too few samples in a local region. Expanding r_m accordingly will ensure that this is always the case. The same effect could possibly be achieved by somehow defining a minimum radius, $r_{\min}$, instead, but is not considered in this thesis.

Define validity functions, ρ_m, by the use of individual smoothing parameters of the form $\lambda_m = \lambda r_m$, where λ is a fixed parameter. This corresponds to point 4 in section 3.3. The result is the same overlap when splitsamples are close, as when they are far from each other.

Algorithm 1b (Variable validity functions)

This algorithm is similar to the original **Algorithm 1a**, as the allocation of training samples is still based on 'hard' splits, by using the same procedure described by equation 4.4. However, the local validity functions are now individually defined.

Once a new splitsample is found, a new local radius, r_{M+1}, is computed. All the other radii affected by the position of the new splitsample are also re-computed. New local validity functions with new smoothing parameters are defined for the regions concerned. A new classification can then take place, based on the largest values of all the validity functions, before new local linear models are computed. The entire algorithm is given in figure 4.5.

1. Preprocessing:
   (a) Removal of possible outliers from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$.
   (b) Training matrices: $\mathbf{X}$ and $\mathbf{y}$ (both assumed to be scaled and centered).
   (c) Test matrices: $\mathbf{X}_{\text{test}}$ and $\mathbf{y}_{\text{test}}$ (both assumed to be scaled and centered with same factors as for the training matrices).
   (d) PCA analysis: Determine A_w from $\mathbf{X}$.
   (e) Projected input matrices: $\mathbf{T}$ and $\mathbf{T}_{\text{test}}$.
2. Initialization:
   (a) Define validity function: $\rho_1 = \exp(-\frac{1}{2}(\mathbf{t}-\mu_1)\Lambda^{-1}(\mathbf{t}-\mu_1)^T)$ where $\mu_1 = \mathbf{0}$, $\Lambda = \lambda_1^2\mathbf{P} = (\lambda r_1)^2\mathbf{P}$, $r_1 = \max_{n=1,\dots,N_{\text{train}}}(\|\mathbf{t}^n - \mu_1\|_{\text{M}})$, and $\mathbf{P}$ is the covariance matrix of $\mathbf{T}$.
   (b) Compute global PLS model $f^{(1)} = f_1$ from $\mathbf{X}$ and $\mathbf{y}$.
   (c) Number of local models, $M = 1$.
3. while <consistent decrease in validation criterion J> do
   (a) Find splitsample: $\mu_{M+1} = \mathbf{t}^{s_M}$ where $s_M = \arg\max_{n=1,\dots,N_{\text{train}}(\neq s_1,\dots,s_{M-1})}(e^n)$ and $e^n = e^n_{(1)}$ or $\bar{e}^n_{(K)}$ or $e^n_{(K)}$ (and $\mathbf{t}^{s_M}$ is consistent).
   (b) Define radius: $r_{M+1} = \min_{m=1,\dots,M}(\|\mu_m - \mu_{M+1}\|_{\text{M}})$
   (c) for radius $m = 1, 2, \dots, M$ do
       - Diminish r_m if $\|\mu_m - \mu_{M+1}\|_{\text{M}} < r_m$.
   (d) for region $m = 1, 2, \dots, M+1$ do
       - Define validity function: $\rho_m = \exp(-\frac{1}{2}(\mathbf{t}-\mu_m)\Lambda^{-1}(\mathbf{t}-\mu_m)^T)$ where $\Lambda = (\lambda r_m)^2\mathbf{P}$.
   (e) for sample $n = 1, 2, \dots, N_{\text{train}}$ do
       - Allocate sample $\mathbf{x}^n$ to $\mathbf{X}_m$ and y^n to $\mathbf{y}_m$ using equation 4.4.
   (f) Local training matrices: $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_{M+1}$ and $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{M+1}$ (all assumed to be centered).
   (g) for region $m = 1, 2, \dots, M+1$ do
       - Compute local PLS model f_m from $\mathbf{X}_m$ and $\mathbf{y}_m$.
   (h) Interpolate: $f^{(M+1)} = \sum_m w_m f_m$.
   (i) Validate $f^{(M+1)}$ using validation criterion $J = \text{RMSEP}(f^{(M+1)}(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}})$.
   (j) Increment: $M = M + 1$.
4. Optimal model: $f = \arg\min_{f^*=f^{(1)},\dots,f^{(M-1)}}(\text{RMSEP}(f^*(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}}))$.

Figure 4.5: **Algorithm 1b** (Variable validity functions).

Algorithm 1c (Overlapping local regions)

This algorithm uses both the new way of allocating samples and individual validity functions.

Once a new splitsample is found, a new local radius, r_{M+1}, is computed, as are all the other radii affected by the position of the new splitsample. Training samples are allocated to the $M+1$ classes using the new allocation procedure described above. During this allocation, local radii could again change in order to comply with the minimum-number-of-samples constraint. The next step is to define new model validity functions and compute new local linear models both for the new region and for those regions, whose corresponding radii have been adjusted since the last iteration. A new total model, $f^{(M+1)}$, can then be validated. The entire algorithm is given in figure 4.6.

Algorithm 1d (Hierarchical local regions)

One drawback with the previous algorithm is that one is *not* guaranteed that each training sample is used in at least one model. This is a waste of samples. **Algorithm 1d** attempts to avoid that by never changing neither the local model nor the validity function nor the radius once they are computed and defined.

All that is done once a new splitsample is found is to compute the new radius, r_{M+1}, define the new validity function, ρ_{M+1}, with parameter λ_{M+1} based on this radius and finally compute the new local model from the training samples within distance r_{M+1} from the center, μ_{M+1}. None of the previous models and model parameters are altered. The new model, f_{M+1}, simply overlaps the old ones. The first global linear model f_1 will then always be at the bottom, with smoothing parameter $\lambda_1 = \lambda r_{\max}$, where $r_{\max}$ is the distance from origo to the most distant training sample. The entire algorithm is given in figure 4.7.

All these three new algorithms were also implemented. The same assumptions as in **Algorithm 1a**, regarding the local PLS modeling, were made (see appendix A).

1. Preprocessing:

   (a) Removal of possible outliers from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$.

   (b) Training matrices: $\mathbf{X}$ and $\mathbf{y}$ (both assumed to be scaled and centered).

   (c) Test matrices: $\mathbf{X}_{\text{test}}$ and $\mathbf{y}_{\text{test}}$ (both assumed to be scaled and centered with same factors as for the training matrices).

   (d) PCA analysis: Determine A_w from $\mathbf{X}$.

   (e) Projected input matrices: $\mathbf{T}$ and $\mathbf{T}_{\text{test}}$.

2. Initialization:

   (a) Define validity function: $\rho_1 = \exp(-\frac{1}{2}(\mathbf{t}-\mu_1)\Lambda^{-1}(\mathbf{t}-\mu_1)^T)$ where $\mu_1 = \mathbf{0}$, $\Lambda = \lambda_1^2\mathbf{P} = (\lambda r_1)^2\mathbf{P}$, $r_1 = \max_{n=1,\ldots,N_{\text{train}}}(\|\mathbf{t}^n - \mu_1\|_{\mathrm{M}})$, and $\mathbf{P}$ is the covariance matrix of $\mathbf{T}$.

   (b) Compute global PLS model $f^{(1)} = f_1$ from $\mathbf{X}$ and $\mathbf{y}$.

   (c) Number of local models, $M = 1$.

3. while <consistent decrease in validation criterion J> do

   (a) Find splitsample: $\mu_{M+1} = \mathbf{t}^{s_M}$ where $s_M = \arg\max_{n=1,\ldots,N_{\text{train}}(\neq s_1,\ldots,s_{M-1})}(e^n)$ and $e^n = e^n_{(1)}$ or $\bar{e}^n_{(K)}$ or $e^n_{(K)}$ (and $\mathbf{t}^{s_M}$ is consistent).

   (b) Define radius: $r_{M+1} = \min_{m=1,\ldots,M}(\|\mu_m - \mu_{M+1}\|_{\mathrm{M}})$

   (c) for radius $m = 1, 2, \ldots, M$ do

      - Diminish r_m if $\|\mu_m - \mu_{M+1}\|_{\mathrm{M}} < r_m$.

   (d) for region $m = 1, 2, \ldots, M+1$ do

      i. for sample $n = 1, 2, \ldots, N_{\text{train}}$ do
         - Allocate sample $\mathbf{x}^n$ to $\mathbf{X}_m$ and y^n to $\mathbf{y}_m$ if $\|\mathbf{t}^n - \mu_m\|_{\mathrm{M}} \leq r_m$.

      ii. Expand r_m if too few samples allocated.

      iii. Define validity function: $\rho_m = \exp(-\frac{1}{2}(\mathbf{t}-\mu_m)\Lambda^{-1}(\mathbf{t}-\mu_m)^T)$ where $\Lambda = (\lambda r_m)^2\mathbf{P}$.

      iv. Compute local PLS model f_m from $\mathbf{X}_m$ and $\mathbf{y}_m$.

   (e) Interpolate: $f^{(M+1)} = \sum_m w_m f_m$.

   (f) Validate $f^{(M+1)}$ using validation criterion $J = \text{RMSEP}(f^{(M+1)}(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}})$.

   (g) Increment: $M = M + 1$.

4. Optimal model: $f = \arg\min_{f^* = f^{(1)},\ldots,f^{(M-1)}}(\text{RMSEP}(f^*(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}}))$.

Figure 4.6: **Algorithm 1c** (Overlapping local regions).

1. Preprocessing:
   (a) Removal of possible outliers from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$.
   (b) Training matrices: $\mathbf{X}$ and $\mathbf{y}$ (both assumed to be scaled and centered).
   (c) Test matrices: $\mathbf{X}_{\text{test}}$ and $\mathbf{y}_{\text{test}}$ (both assumed to be scaled and centered with same factors as for the training matrices).
   (d) PCA analysis: Determine A_w from $\mathbf{X}$.
   (e) Projected input matrices: $\mathbf{T}$ and $\mathbf{T}_{\text{test}}$.
2. Initialization:
   (a) Define validity function: $\rho_1 = \exp(-\frac{1}{2}(\mathbf{t}-\mu_1)\Lambda^{-1}(\mathbf{t}-\mu_1)^T)$ where $\mu_1 = \mathbf{0}$, $\Lambda = \lambda_1^2\mathbf{P} = (\lambda r_1)^2\mathbf{P}$, $r_1 = \max_{n=1,\dots,N_{\text{train}}}(\|\mathbf{t}^n - \mu_1\|_{\text{M}})$, and $\mathbf{P}$ is the covariance matrix of $\mathbf{T}$.
   (b) Compute global PLS model $f^{(1)} = f_1$ from $\mathbf{X}$ and $\mathbf{y}$.
   (c) Number of local models, $M = 1$.
3. while $<$consistent decrease in validation criterion $J>$ do
   (a) Find splitsample: $\mu_{M+1} = \mathbf{t}^{s_M}$ where $s_M = \arg\max_{n=1,\dots,N_{\text{train}}(\neq s_1,\dots,s_{M-1})}(e^n)$ and $e^n = e^n_{(1)}$ or $\bar{e}^n_{(K)}$ or $e^n_{(K)}$ (and $\mathbf{t}^{s_M}$ is consistent).
   (b) Define radius: $r_{M+1} = \min_{m=1,\dots,M}(\|\mu_m - \mu_{M+1}\|_{\text{M}})$
   (c) for sample $n = 1, 2, \dots, N_{\text{train}}$ do
      - Allocate sample $\mathbf{x}^n$ to $\mathbf{X}_{M+1}$ and y^n to $\mathbf{y}_{M+1}$ if $\|\mathbf{t}^n - \mu_{M+1}\|_{\text{M}} \leq r_{M+1}$.
   (d) Expand r_{M+1} if too few samples allocated.
   (e) Define validity function: $\rho_{M+1} = \exp(-\frac{1}{2}(\mathbf{t} - \mu_{M+1})\Lambda^{-1}$ $(\mathbf{t} - \mu_{M+1})^T)$ where $\Lambda = (\lambda r_{M+1})^2\mathbf{P}$.
   (f) Compute local PLS model f_{M+1} from $\mathbf{X}_{M+1}$ and $\mathbf{y}_{M+1}$.
   (g) Interpolate: $f^{(M+1)} = \sum_m w_m f_m$.
   (h) Validate $f^{(M+1)}$ using validation criterion $J = \text{RMSEP}(f^{(M+1)}(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}})$.
   (i) Increment: $M = M + 1$.
4. Optimal model: $f = \arg\min_{f^*=f^{(1)},\dots,f^{(M-1)}}(\text{RMSEP}(f^*(\mathbf{X}_{\text{test}}), \mathbf{y}_{\text{test}}))$.

Figure 4.7: **Algorithm 1d** (Hierarchical local regions).

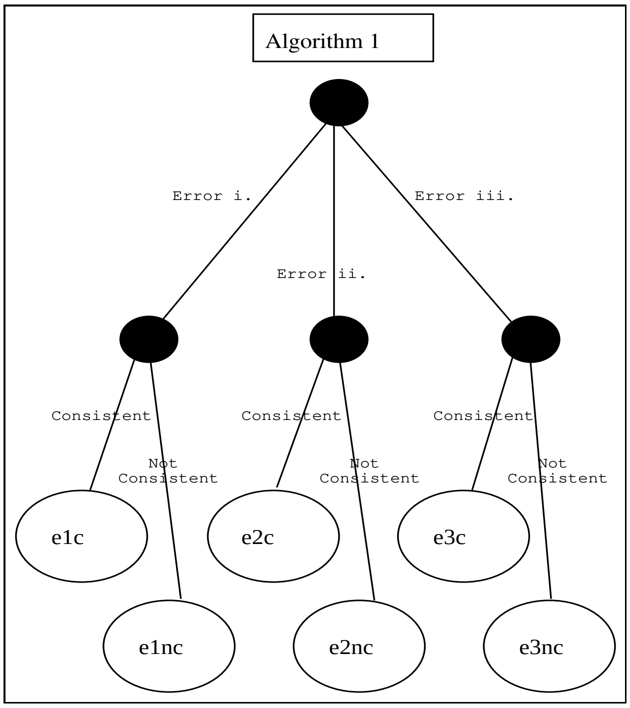

Figure 4.8: The different combinations of error measure and heuristic.

4.4 Evaluation on simple test examples

One of the difficulties with all the four versions of **Algorithm 1** is the large number of different combinations of error measure and heuristic that are possible. The total number is 6, as illustrated in figure 4.8. In addition, the value of the external parameters K, the number of neighboring samples, λ, the degree of overlap, and A_w, the number of principal components, must be determined for each combination. The procedure for finding the optimal combination is therefore very computationally demanding.

To examine the behavior of the four algorithms, they were all tested on a few simple examples. These examples are described in the next section, together with the results of **Algorithm 1a**. The most important results of the three other algorithms are given in section 4.4.2.

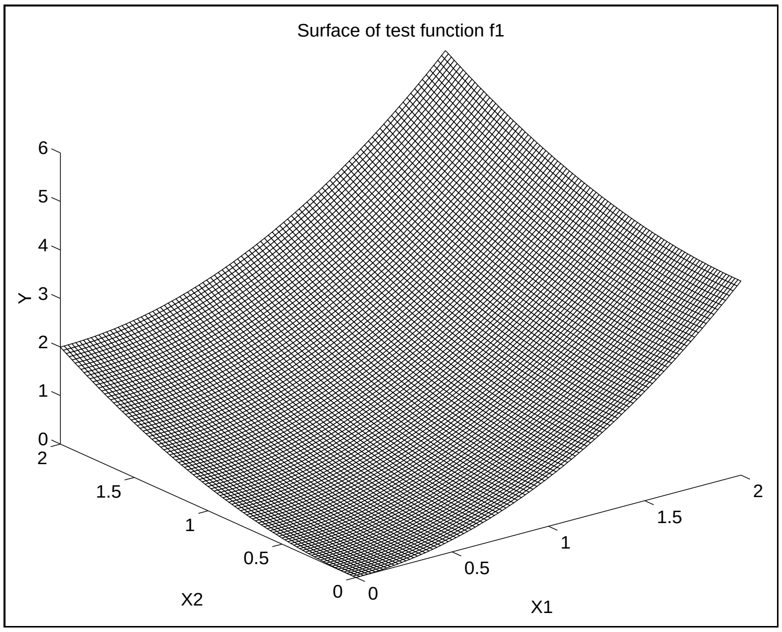

Figure 4.9: Surface of f_1.

4.4.1 Algorithm 1a

To investigate the properties of **Algorithm 1a**, the algorithm was first tested on the smooth and simple low dimensional ($P = 2$) function

$$y = f_1(x_1, x_2) = (x_1)^2 + \frac{1}{2}(x_2)^2 \tag{4.6}$$

whose surface is illustrated in figure 4.9.

The goal of this simple nonlinear example was just to illustrate that the principle of local modeling is sensible, and that the algorithm worked. The number of input variables was chosen to be 2, which provides simple three-dimensional plots of the relationship between the variables.

All the samples of x_1 and x_2 were randomly drawn from a uniform [0,2] distribution. Two training sets of 100 samples each and one test set of 500 samples were constructed. One of the training sets was without noise on the output samples. To the y^n's in the other set, white noise, $e \sim N(0, \sigma_e^2)$ with $\sigma_e = 0.5$, corresponding to ≈ 37 % ($\sigma_e/S(\mathbf{y})$) noise level, was added. Noise was not added to the test set.

The same procedure was then repeated, but the number of samples in the two training sets was now 300. In the test set it was still 500, though. The purpose was to investigate what happens when the number of training samples increases. In all the tables the two noiseless training sets are named 100 and 300, whereas the two noisy sets are named 100n and 300n.

Model	100	100n	300	300n
Linear	0.324	0.328	0.347	0.339
Quadratic	0.000	0.084	0.000	0.055
e1c	0.089 (7)	0.203 (5) *	0.061 (9)	0.177 (3)
e1nc	0.089 (7)	0.195 (3)	0.061 (9)	0.128 (8)
e2c	0.092 (7)	0.201 (5) *	0.058 (9)	0.145 (6)
e2nc	0.065 (9)	0.176 (7)	0.058 (9)	0.145 (6)
e3c	0.101 (6)	0.209 (6) *	0.053 (10)	0.145 (6)
e3nc	0.101 (6)	0.194 (4)	0.053 (10)	0.155 (8)

Table 4.1: RMSEP for different models for approximating f_1[a].

Algorithm 1a was then run with fixed $K = 5$ and $\lambda = \frac{1}{2}$ for the four different data sets and involving all 6 combinations. The results in RMSEP are given in table 4.1. First of all, this table clearly shows, as was intended, that local linear modeling indeed decreases the prediction error, compared to what a global linear model does. This hardly comes as a surprise. Secondly, one observes that the generalization properties are better, both for noiseless and noisy data sets, with 300 training samples than with only 100. Again, a rather obvious result, since an increase in the number of samples will allow more local models and also 'even out' the influence of noise in the samples. The third observation is that the prediction error is larger for the noisy data sets than for the noiseless, when the same test set is used. Something, which of course is as expected.

What is more interesting though, is to look at the estimation error. For noiseless data, the RMSEE had approximately the same evolution, with regard to the order of magnitude, as the RMSEP when M increased. For noisy data, the RMSEE was larger than RMSEP, but the relative evolution was still the same. With 100 training samples, the RMSEE was reduced from 0.604 (linear model) to 0.46–0.48 (best local linear models), and with 300 samples from 0.605 to 0.50–0.52. This corresponds to the noise level (0.5), and indicates that no further improvement can be expected since only noise is left to be modelled. Thus, the algorithm is able to withdraw as much information as possible from the noisy data, which is an encouraging property. Another way of showing this is to compute the RMSE between the noisy output training samples and the corresponding 'true' output samples. This value was also approximately 0.5 for the best local linear models.

The best noiseless local linear model was *e3nc*, and the best based on noisy samples was *e1nc*. The model surfaces are illustrated in figure 4.10 and figure 4.11 respectively. These figures can be compared with figure 4.9.

[a] Number of local models is given in parenthesis. An asterisk (*) means that the algorithm aborted when no consistent sample was found, whereas a hash mark (#) means that it aborted due to fewer than 4 samples in a local region.

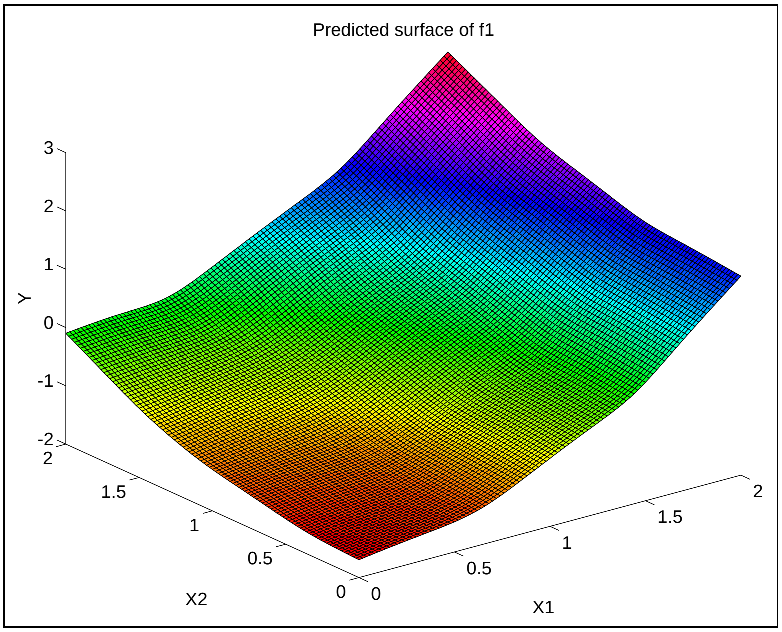

Figure 4.10: Predicted surface of f_1 based on 300 training samples without noise and 10 local models.

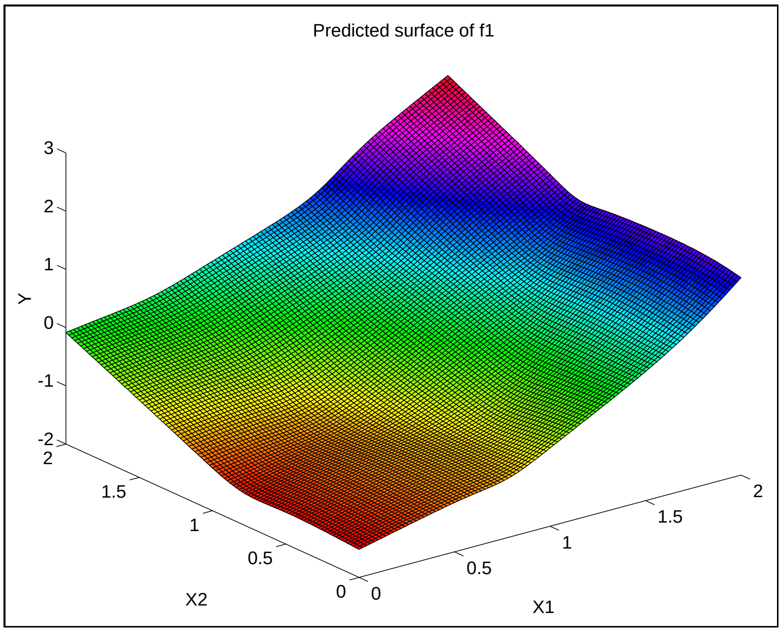

Figure 4.11: Predicted surface of f_1 based on 300 noisy training samples and 8 local models.

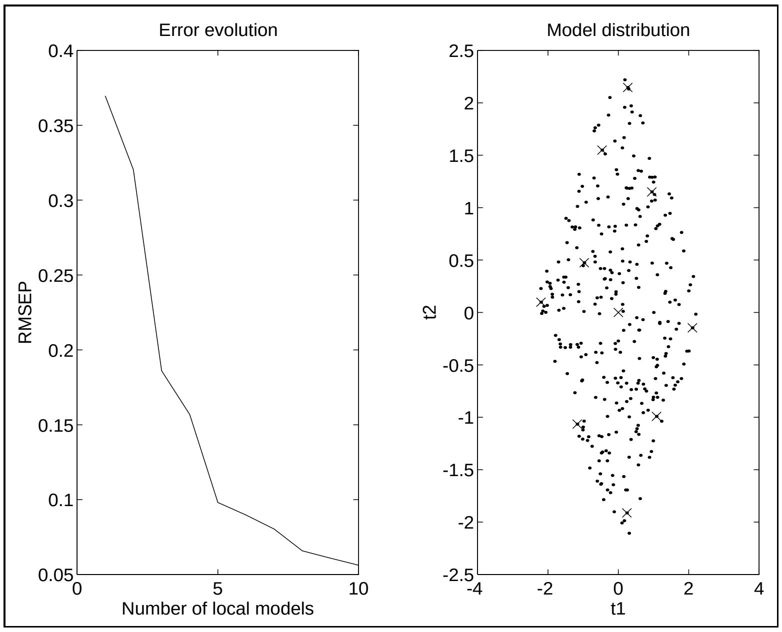

Figure 4.12: The model with lowest minimum RMSEP based on 300 training samples without noise. (a) Evolution of RMSEP. (b) Distribution of local centers (x) in weighting space.

The evolution of the RMSEP as M increases for the same two models is given in figure 4.12a and figure 4.13a. These curves are typical of the general behavior of the RMSEP. Often, at some stage, it slightly rises before decreasing again in the next step. The illustrations on the right in these figures show the distribution of the local centers μ_m, i.e. splitsamples, in W. For a simple homogeneous function like f_1, it is the best if they are as evenly spread out in the input space as possible. As seen from the illustrations this was indeed the case, which is another sensible property of **Algorithm 1a**.

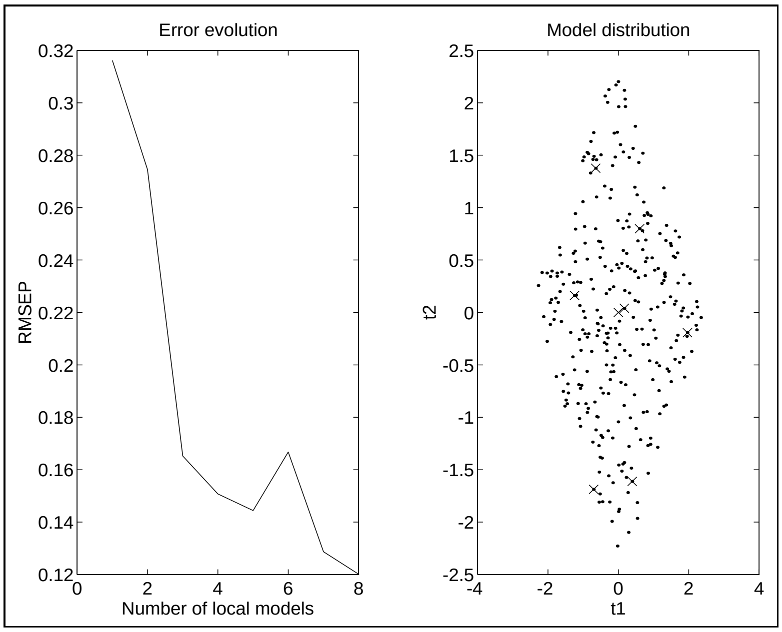

Figure 4.13: The model with lowest minimum RMSEP based on 300 noisy training samples. (a) Evolution of RMSEP. (b) Distribution of local centers (x) in weighting space.

Characteristic	100	100n	300	300n
Error i.	0.089	0.200	0.061	0.155
Error ii.	0.080	0.189	0.058	0.145
Error iii.	0.101	0.202	0.053	0.150
Consistent	0.094	0.205	0.057	0.156
Not consistent	0.087	0.188	0.057	0.143

Table 4.2: RMSEP for different characteristics.

None of the 6 combinations of error measure and heuristic was superior to the others. However, to still be able to possibly disregard some of the combinations, the RMSEP for the results in table 4.1 averaged over the different characteristics, were computed. The following general interpretations were made based on table 4.2:

- The differences between the sample errors were only marginal. No conclusions can be made based on this example only.
- Requiring that the splitsample should be consistent worked really bad if there were few training samples and the noise level was high. The condition was simply too strong, at least for $K = 5$.

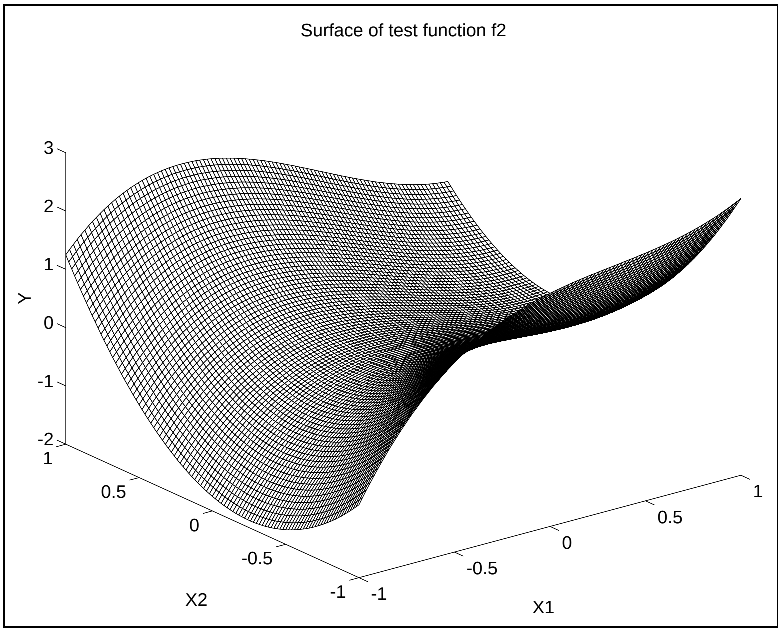

Figure 4.14: Surface of f_2.

The values of K and λ were both fixed in the previous testing. However, to investigate the consequences of varying values, K was first increased from 5 via 10 to 20. The result was a more stable RMSEP evolution, although the minimum was at the same level. Then, the consistency heuristic, with $K = 3$, was tested. Again, no improvements of the best results in table 4.1 were observed. A large K will almost replace the consistency heuristic, since the mean and median sample error will be computed over a larger region. The drawback is that computation of the nearest neighboring samples will be slower. The value of λ was also changed from $\frac{1}{2}$ to both larger and smaller values. However, the effect was only reduced prediction ability.

Algorithm 1a was then tested on a much more nonlinear function

$$y = f_2(x_1, x_2) = (x_1)^3 - (x_1)^2 - \frac{1}{4}x_1 - x_1x_2 + 2(x_2)^2 \tag{4.7}$$

whose surface is illustrated in figure 4.14.

The goal of this example was to illustrate that the algorithm is sensible also on highly nonlinear problems.

All the samples of x_1 and x_2 were now drawn from a uniform [-1,1] distribution. Again a noiseless test set of 500 samples and two training sets of 200 samples each, were constructed. White noise, $e \sim N(0, \sigma_e^2)$ with $\sigma_e = 0.5$, was added to the output samples in one of the training sets. This corresponds to a noise level ($\sigma_e/S(\mathbf{y})$) of 63 %, which is much, perhaps too much. The noiseless training set is named 200, and the noisy set 200n in all the tables.

Model	200	200n
Linear	0.764	0.762
Quadratic (x)	0.159	0.184
Cubic (x)	0.000	0.116
e1c	0.134 (8) #	0.438 (3) *
e1nc	0.120 (8)	0.221 (6)
e2c	0.170 (10) #	0.435 (3) *
e2nc	0.170 (10) #	0.201 (6)
e3c	0.169 (12) #	0.385 (3) *
e3nc	0.161 (12)	0.203 (6)

Table 4.3: RMSEP for different models for approximating f_2[b].

The lowest RMSEP for different models are given in table 4.3, for fixed $K = 10$ and $\lambda = \frac{1}{2}$. Generally, local modeling vastly improved the prediction results compared to a linear model, and also fared well against higher order polynomial models with cross-terms.

For the noiseless data set, the best local linear models were those using the sample error of type $i.$. Probably because without noise on the training samples, there will be no outliers, thus the single error approach will give a correct and undisturbed reflection of where the prediction is worst. Even though, the contrast to the other error measures was still not significant.

The number of local models was relatively large, and would have been even greater had the training set been larger. As it was now, the algorithm aborted for almost every combination. This because too few training samples were allocated to one class, but without having reached the minimum RMSEP, indicating that additional improvement would have been possible.

Figure 4.15 illustrates the model surface of the best local linear model, *e1nc*. If this surface is compared to the original surface (figure 4.14), one observes that although there are differences, the general features of f_2 are captured, so the approximation was not that bad.

[b]See table 4.1 for explanation of symbols.

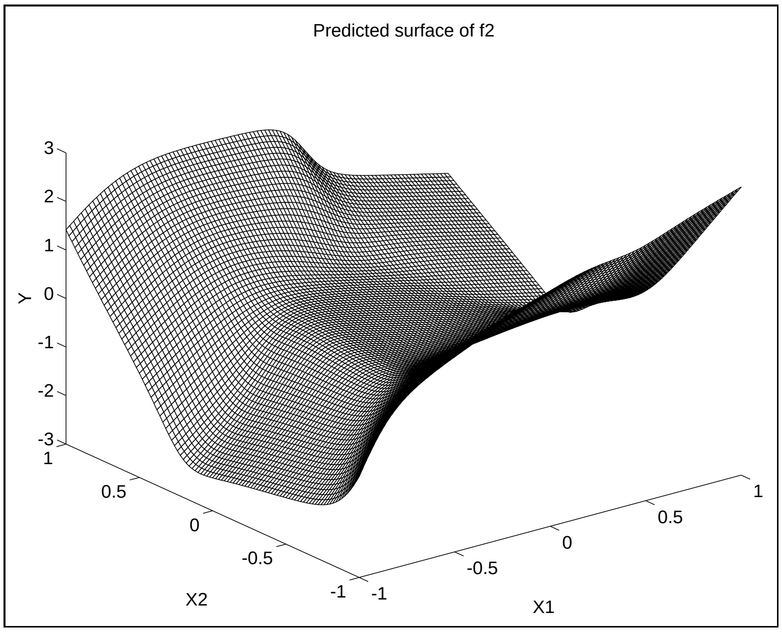

Figure 4.15: Predicted surface of f_2 based on training samples without noise and 8 local models.

The evolution of the RMSEP and the distribution of the local centers μ_m in W, for the same model, are given in figure 4.16. Again, the local centers were well spread out in the input space. This can be seen even better from figure 4.17, where the weight, $w_m(\mathbf{x})$, of each of the local models is illustrated as a function of input space position. A comparison between the peaks in this illustration and the position of the local centers, although rotated and stretched by the PCA, indicates that even though two splitsamples have an almost identical position, the corresponding weight functions will peak apart from each other because they are normalized.

The drawback with the noisy data set was that it was so noisy that the algorithm aborted quickly if consistent splitsamples were required. Using the heuristic was therefore impossible, at least for $K = 10$. The number of neighboring samples was then reduced to 5, which resulted in no abortions, and prediction results almost on the same level as those obtained without the heuristic in table 4.3.

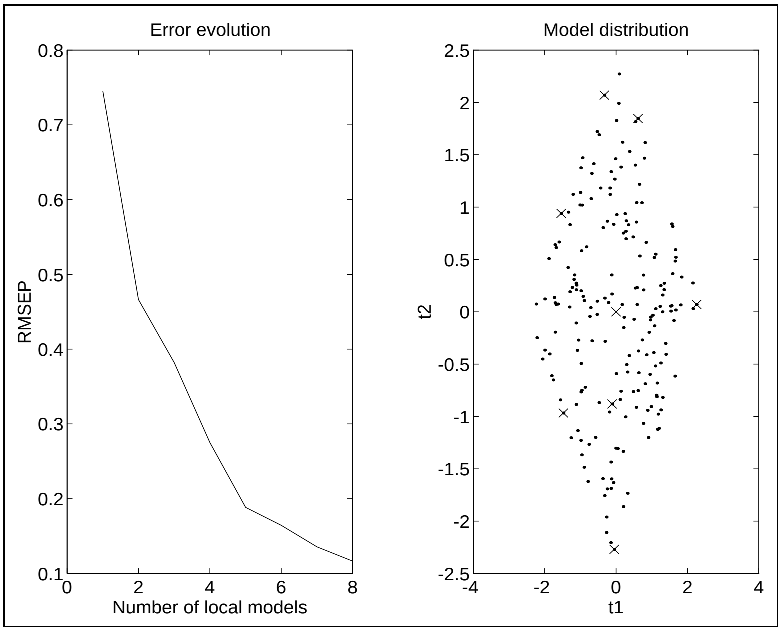

Figure 4.16: The model with lowest minimum RMSEP based on training samples without noise. (a) Evolution of RMSEP. (b) Distribution of local centers (x) in weighting space.

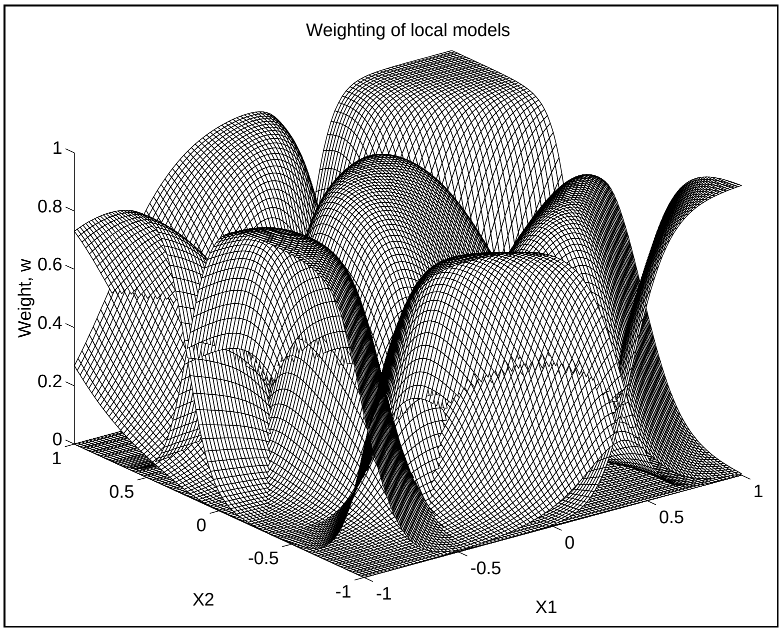

Figure 4.17: Weighting of different local models in the model with lowest minimum RMSEP.

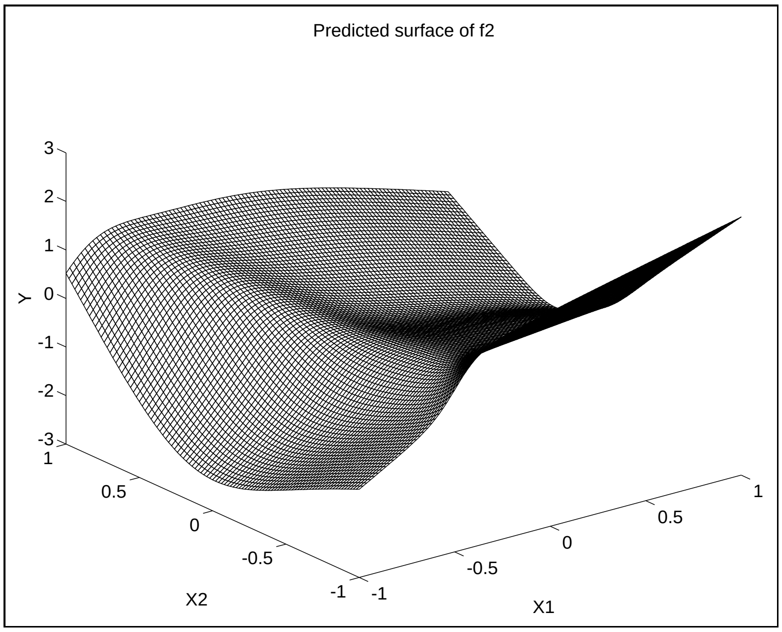

Figure 4.18: Predicted surface of f_2 based on noisy training samples and 6 local models.

When the algorithm did not pre-terminate, inspection of the RMSEE showed once more that the noise level of 0.5 was reached at minimum RMSEP. The RMSEE was reduced from 0.891 (linear model) to 0.50-0.52 (best local linear models). Although the data set was very noisy, the model surface of the best local linear model, *e2nc*, is given in figure 4.18 for completeness.

The most important conclusion that could be drawn from testing **Algorithm 1a** on these two small examples was:

- The algorithm was a sensible approach in low dimensional problems.

Model	Data set	RMSEP
e2nc	100	0.0600 (9)
e2nc	100n	0.263 (4) #
e3nc	100n	0.184 (3) #
e3nc	300	0.046 (10)
e1nc	300n	0.162 (6) #
e3nc	300n	0.221 (3) #
e1nc	200	0.092 (12)
e1nc	200n	0.259 (6) #
e2nc	200n	0.318 (7) #
e3nc	200n	0.411 (3) #

Table 4.4: RMSEP for different models using **Algorithm 1b**[c].

4.4.2 Algorithm 1b, 1c, and 1d

The three other algorithms were then tested on the same two-dimensional functions defined in the previous section, but not quite as thoroughly. Concentration was on the noisy data sets, and only the best combinations of error measure and heuristic in table 4.1 and table 4.3 were considered. The goal was to investigate whether any of the algorithms **1b**,**1c** or **1d** worked better than **Algorithm 1a**. Of the external parameters, K was the same as before, whereas λ was slightly smaller to harmonize with the new definition.

Algorithm 1b

The performance of this algorithm was very similar to that of **Algorithm 1a**, as can be seen from comparing the small collection of results given in table 4.4 with those previously given in tables 4.1 and 4.3. Note that for this algorithm λ was now $\frac{1}{3}$. The only problem was a larger tendency of the algorithm aborting because of regions with too few samples. Otherwise, the change to individual local validity functions did not seem to have that much impact on the prediction ability.

[c]See table 4.1 for explanation of symbols.

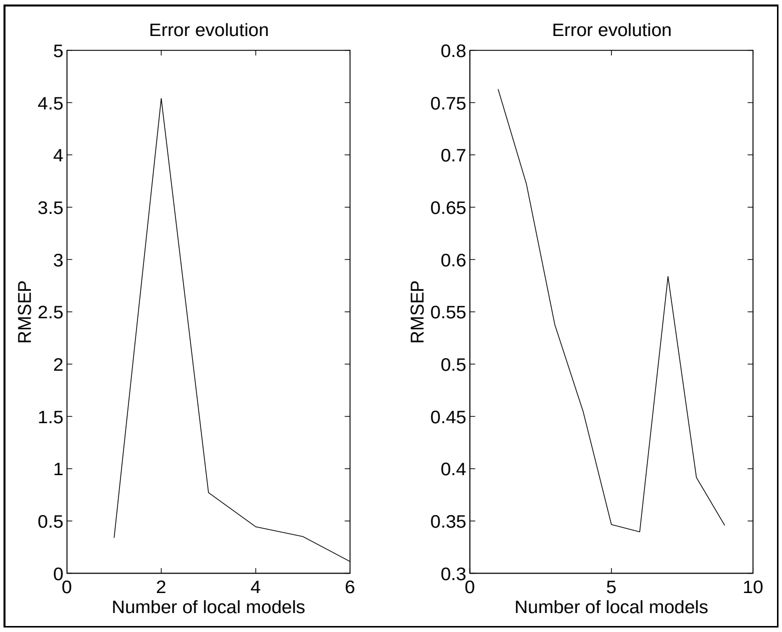

Figure 4.19: Large jumps in the evolution of the RMSEP for **Algorithm 1c**. (a) 300n. (b) 200n.

Algorithm 1c

The prediction ability was generally on the same level as for the first algorithm. A little better for f_1, but worse for the second function. However, the major drawback was the large increase in RMSEP which occurs when the position of a new splitsample forces large regions to diminish, because of the reduction of the radius in these regions. The corresponding local models are then computed from far less samples than in the previous iteration. Typical examples are given in figure 4.19. This behavior makes the algorithm very unstable and accidental, and not very robust, although the minimum RMSEP compared well to that of **Algorithm 1a**.

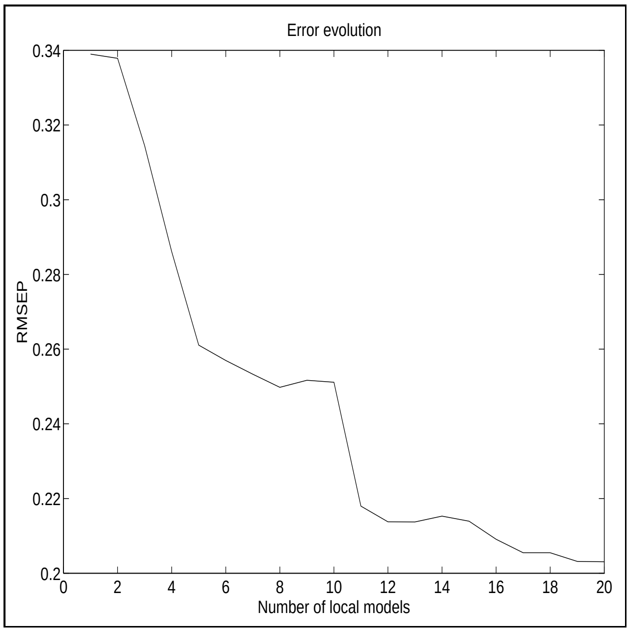

Figure 4.20: Typical example of prediction error evolution for **Algorithm 1d** (300n).

Algorithm 1d

This algorithm did not work very well. The prediction ability was generally much worse than that of **Algorithm 1a**, with the minimum RMSEP being consistently 50–75% higher. A typical example of the evolution of the RMSEP is given in figure 4.20. From this illustration, one sees that despite the very high number of local models, the performance is not very good even though the curve is still slowly decreasing. For all the other algorithms, the minimum RMSEP for this particular example was at least less than 0.15. The main reason being that in this algorithm the influence from the first global linear model will always be too strong. Many of the other models will also be very local (based on the minimum number of samples only) and with very limited validity.

The general conclusions that could be drawn from testing these three algorithms on the small low dimensional examples were:

- None of the algorithms clearly outperformed the original **Algorithm 1a**.
- **Algorithm 1c** is dropped, because of the instability when not all the training samples were used in an iteration.
- **Algorithm 1d** is disregarded, because it performed significantly worse than all the other.

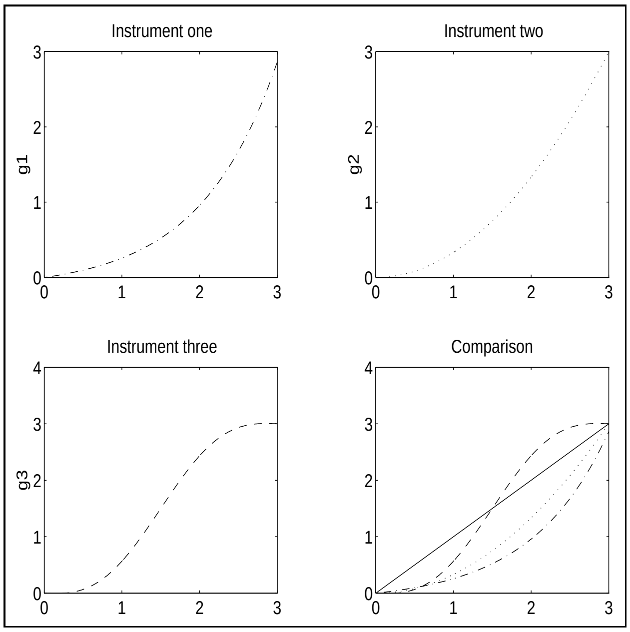

Figure 4.21: Nonlinear response functions for the three instruments.

4.5 Evaluation on a more complex test example

So far the algorithms have been tested only on two-dimensional data sets. Neither the concept of weighting in a projected subspace nor the local PLS approach were then actually tested, since a full PLS solution with two latent variables, i.e. equivalent to the MLR solution, was used in each local model. In addition, there was no correlation between the input variables since they were independently generated.

To further investigate the properties, a high dimensional data set with 30 input variables was generated. The artificial set is based on a nonlinear mixture model from NIR spectroscopy [Martens and Næs 91], where the following situation is simulated:

Three compounds (latent variables) are mixed together in one chemical solution. The measurements (input variables) are divided into three categories corresponding to three different instruments, each measuring the absorbance spectra of the mixture at 10 different frequencies. Each instrument has a different nonlinear response, as illustrated in figure 4.21, with $g_1(z) = 0.15(\exp(z) - 1)$, $g_2(z) = \frac{1}{3}z^2$ and $g_3(z) = z - 0.5\sin(\frac{2}{3}\pi z)$. One element in the input matrix, $\mathbf{X}$, is then of the form

$$\mathbf{X}_{n,j+10(i-1)} = g_i((\mathbf{QA})_{n,j+10(i-1)}), i = 1, 2, 3, j = 1, 2, ..., 10 \tag{4.8}$$

where $\mathbf{Q}$ is a $N \times 3$ matrix whose general element $\mathbf{Q}_{n,k}$ is the concentration of substance k in sample n, and $\mathbf{A}$ is a 3×30 matrix where $\mathbf{A}_{k,j+10(i-1)}$ is the coefficient of absorbance corresponding to compound k at frequency number j for instrument i.

Model	**Algorithm 1a**	**Algorithm 1b**
Linear PLS (5 lv)	0.055	
MLR (=PLS (30 lv))	0.057	
Quadratic PLS (10 lv)	0.036	
Quadratic (60 lv)	0.039	
Cubic PLS (17 lv)	0.030	
Cubic (90 lv)	0.045	
e1nc	0.030 (6)	0.037 (7)
e1c	0.038 (4)	0.032 (5)
e2nc	0.023 (6)	0.031 (7)
e2c	0.030 (6)	0.031 (7)
e3nc	0.034 (5)	0.038 (3)
e3c	0.038 (4)	0.034 (4)

Table 4.5: RMSEP for different models for the spectrometry data set[d].

The output variable is simply one of the compounds. Which one, is arbitrary since all three are generated in the same way. The one, where the RMSEP and RMSEE for an initial linear PLS model were largest, indicating most potential for improvement, was chosen. This was number three, i.e. $\mathbf{y}_n = \mathbf{Q}_{n,3}$.

A training set of 150 and a test set of 50 samples were generated. All the elements in both **Q** and **A** were randomly drawn from a uniform [0,1] distribution. In addition, 10 % white noise was added to the 30 input variables in both the training and test set.

Both **Algorithm 1a** and **1b** were first tested with $\lambda = \frac{1}{2}$ and $K = 5$. The number of principal components, A_w, in the weighting subspace W was 3, equaling the number of compounds. The results in RMSEP are given in table 4.5. In the local modeling, the number of latent variables in each local PLS model varied from 3 to 6.

The results clearly indicate that both local modeling algorithms improve the prediction, with the lowest RMSEP being under half that of the linear PLS model. The main reason is the nonlinearity in the data, which the linear PLS model was not able to model particularly well, as seen from the slight curvature in figure 4.22. With local modeling this curvature is no longer present (figure 4.23). Local modeling was also superior to 2nd and 3rd order PLS (without cross terms).

[d]See table 4.1 for explanation of symbols.

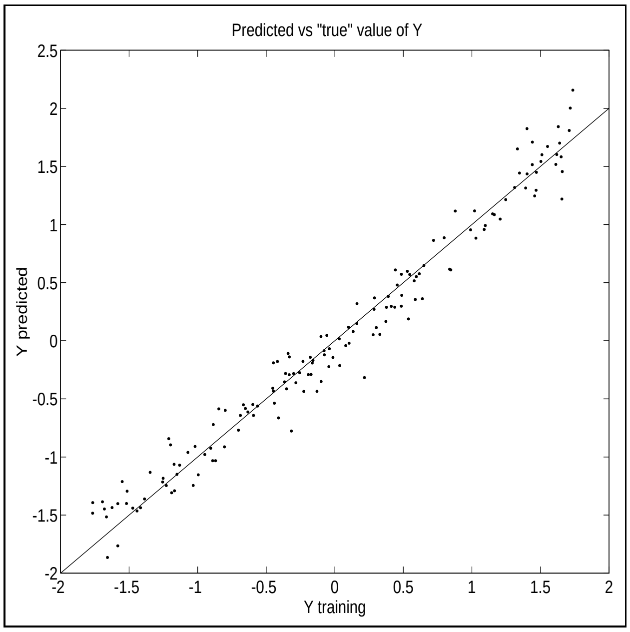

Figure 4.22: Estimation plot for linear PLS model.

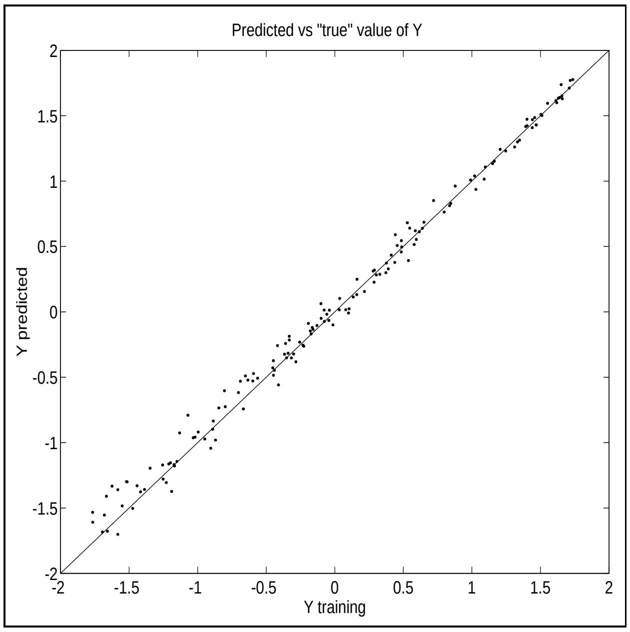

Figure 4.23: Estimation plot for local model with lowest minimum RMSEP.

For **Algorithm 1a**, λ was changed to $\frac{1}{4}$, $\frac{3}{4}$, and 1 without improving the results at all. For $\lambda = \frac{1}{2}$, $K = 3$ and $K = 10$ was also tested. Again the results were generally worse compared to those in table 4.5. For **Algorithm 1b**, a change in the value of λ to $\frac{3}{4}$ resulted in much higher RMSEP, whereas the results for $\lambda = \frac{1}{3}$ and $\frac{1}{4}$ were generally on the same level as those in the table. This is to be expected since the radii r_m, due to the way of scaling the axes in W, are very often larger than 1 which corresponds to the fixed validity function used in **Algorithm 1a**. The optimal value of λ will then be smaller for **Algorithm 1b** than for the first algorithm.

One difference between the two algorithms was that overfitting was more of a problem using **Algorithm 1b** than **Algorithm 1a**. Generally, the RMSEE was lower, but the RMSEP was higher for the former algorithm compared to the latter one. Thus, **Algorithm 1b** is able to better approximate the training samples, probably because of the use of individual validity functions. It must be said though, that the difference was not much. The RMSEE was reduced from 0.049 (linear PLS) to 0.020-0.024 in both cases.

The average correlation (see appendix B.1) between the different input variables was 0.81, which indicates strongly correlated variables. In comparison the average correlation for the artificial data sets generated from f_1 and f_2 was between 0.01 and 0.12.

The main conclusions that could be drawn from this high dimensional problem with correlated input variables were:

- Both local modeling algorithms worked much better than linear PLS.
- The concept of low dimensional weighting space and local PLS modeling seemed fruitful even when P was large.

4.6 Other algorithms and modifications

The general **Algorithm 1** is a local search algorithm which will lead to a locally suboptimal solution. It is best implemented as an iterative *depth-first* search. Since only one new model decomposition is investigated in each iteration step, **Algorithm 1** is completely based on the assumption that improving the worst case behavior of $f^{(M)}$ also improves the prediction ability the most. However, there is no guarantee of that.

In order to be less dependent on this assumption, two natural generalizations of **Algorithm 1** are given below. Both give more optimal solutions.

Algorithm 2

The globally optimal solution is obtained if *all* the possible decompositions are examined, and not only the ones indirectly specified by the splitsamples as in **Algorithm 1**. In **Algorithm 2**, the first two steps are not any different from those in the first algorithm. Steps 3 and 4 are, however, replaced by the following steps:

First define $M_{\max}$, the maximum number of local models. Then, use stepwise decomposition where at each step $M = 2, 3..., M_{\max}$, all, but the previously used, samples in $\mathcal{D}_{\text{train}}$ are selected as splitsamples for new local models. For each decomposition compute new local linear models, interpolate them, and validate the total model. The optimal model is the one that minimizes the validation criterion J.

This algorithm performs a global search, which is best implemented in a combinatorial way. Each decomposition is based on the same principles as in **Algorithm 1**. The total number of possible, not necessarily different, decompositions at step M is

$$(N_{\text{train}})(N_{\text{train}} - 1) \cdots (N_{\text{train}} + 2 - M) = \frac{(N_{\text{train}})!}{(N_{\text{train}} + 1 - M)!} \tag{4.9}$$

This situation is illustrated in figure 4.24, where each node in the tree corresponds to a decomposition, and the numbers in a node refer to the splitsample indices, $s_m, m = 1, ..., M - 1$. In **Algorithm 2** all the nodes are investigated, as opposed to the first algorithm which only investigates the nodes along one path. **Algorithm 2** is therefore completely independent of the assumption about a connection between the largest sample error and the largest improvement in prediction ability.

Unfortunately, this algorithm will be *very* computationally demanding even for small values of N_{train} and $M_{\max}$. It is, therefore, practically impossible to apply without including heuristics, which drastically reduce the computation time. One such heuristic is to stop the decomposition in a branch if the next splitsample is not *consistent*. Another is to stop if a new decomposition *increased* the RMSEE. Using such heuristics will give suboptimal, but often good enough solutions.

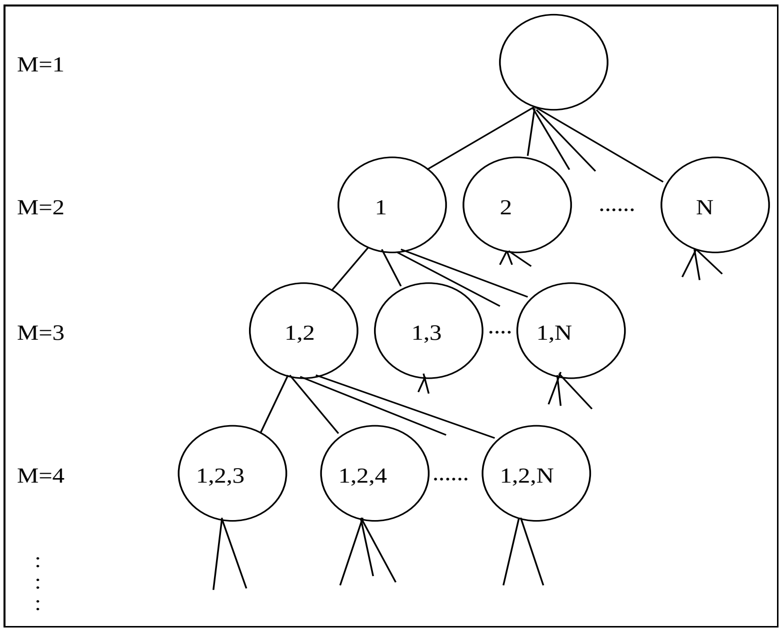

Figure 4.24: Possible decompositions based on the training samples.

Algorithm 3

This algorithm lies somewhere between the other two, since an extended local search is performed. Steps 1, 2, and 4 are all unaltered, but the repeated step (3) in **Algorithm 1** is now changed as follows:

First choose the L samples, $1 \leq L \leq N_{\text{train}} + 1 - M$*, of all the* $N_{\text{train}} + 1 - M$ *non-selected samples having the largest sample error of type* ***i.***, ***ii.***, *or* ***iii.*** *as candidates for the splitsample. For all these candidates, decompose the input space, compute new local linear models and interpolate them. Select, as splitsample, the one of the candidates whose decomposition and corresponding total model,* $f_l^{(M+1)}$*, gives the smallest RMSEE. Continue with that total model,* $f^{(M+1)} = f_l^{(M+1)}$*, in the validation.*

Algorithm 3 is less sensitive to the assumption about prediction improvement. This because the decompositions from the L largest sample errors, and not only *the* largest, are investigated in each iteration step. Therefore, the algorithm will be more computationally demanding than **Algorithm 1**, but obviously faster than **Algorithm 2**. Again, heuristics such as requiring *consistent* candidate samples will speed up the algorithm. As the first algorithm, **Algorithm 3** is best implemented as an iterative *depth-first* search.

In the limit $L = N_{\text{train}} + 1 - M$, the solution is one-step-ahead optimal in the sense that it is optimal in that iteration step, since all possible decompositions are considered. In figure 4.24 that corresponds to looking at all the nodes in the next level of a subtree, before deciding which node to enter. In the other special case $L = 1$, **Algorithm 3** is simply reduced to **Algorithm 1**.

Although both of these two generalizations, of the first algorithm, most likely will give slightly better solutions in terms of prediction ability, they were not implemented because of their much higher computational demands. In the rest of this thesis all additional testing and discussion is therefore related to **Algorithm 1**. The other two algorithms are not considered any further.

A modification of the strategy for determining the splitsample is then proposed.

Strategy 2

First compute the different local RMSEE of the M local regions, using only local samples in the computation. Define the region with the largest local RMSEE as the region of splitting. *Select the splitsample as the sample having the largest sample error of type* ***i.***,***ii.*** *or* ***iii.***, *under the condition that this sample is located within the region of splitting.*

This strategy can be seen as a local version of strategy 1, with the search for a splitsample restricted to a local region. However, the main purpose is to preserve an as equal as possible degree of local linearity in the different regions, measured in terms of local RMSEE. This principle gives a total model, f, with an approximately equally good prediction ability everywhere in the input hyperspace, which is often desirable.

To investigate this strategy, **Algorithm 1a** was again tested on the two low dimensional functions, but now with strategy 2 as the way of determining the splitsample. The results are given in table 4.6 and 4.7.

Model	100	100n	300	300n
e1c	0.122 (6) *	0.303 (3) *	0.062 (9)	0.174 (6) *
e1nc	0.122 (6) #	0.195 (3)	0.062 (9)	0.133 (5)
e2c	0.085 (7)	0.215 (4) *	0.061 (9)	0.240 (5) *
e2nc	0.061 (9)	0.187 (7) #	0.061 (9)	0.159 (8)
e3c	0.101 (6)	0.239 (3) *	0.063 (8)	0.174 (3)
e3nc	0.101 (6)	0.187 (5) #	0.063 (8)	0.143 (7)

Table 4.6: RMSEP for different models for approximating f_1 (strategy 2)[e].

[e]See table 4.1 for explanation of symbols.

Model	200	200n
e1c	0.180 (6) #	0.438 (3) *
e1nc	0.141 (8) #	0.259 (5)
e2c	0.142 (12) #	0.435 (3) *
e2nc	0.142 (12) #	0.276 (4)
e3c	0.177 (9) #	0.385 (3) *
e3nc	0.170 (12)	0.291 (5) #

Table 4.7: RMSEP for different models for approximating f_2 (strategy 2)[f].

Characteristic	100	100n	300	300n	200	200n
Strategy 1	0.091	0.197	0.057	0.150	0.155	0.332
Strategy 2	0.101	0.225	0.062	0.174	0.160	0.355

Table 4.8: RMSEP for different strategies.

These tables correspond to table 4.1 and 4.3 for strategy 1. To be better able to compare the two strategies, the RMSEP for the results in the four tables averaged over the strategies were computed.

As seen from table 4.8 there was only one conclusion:

- Strategy 2 was worse than strategy 1, and is therefore disregarded.

[f] See table 4.1 for explanation of symbols.

5

Case studies

This chapter describes a set of case studies that have been carried out in order to test the new local modeling algorithms on well-known data sets, and to compare them with other modeling techniques.

5.1 Introduction

So far the algorithms and their properties have only been investigated using specially designed and artificially generated data sets. But do they behave any differently when tested on real world examples? Another important question is, how well do the local modeling algorithms perform compared to other nonlinear modeling techniques? These two questions will be addressed in this chapter.

Comparisons between different modeling techniques and algorithms are always difficult, as no technique will constantly outperform all the others on whatever data set imaginable. Usually, algorithms are designed for a special modeling purpose and will work satisfactory on these kind of problems. A good example is PLS, which works very well in NIR (near infrared) spectroscopy where the number of input variables (i.e. wavelengths) is very high (often $P > 100$), the size of the training set is small, and the variables are highly correlated. On more lower dimensional and larger data sets though, other modeling techniques may perform better than PLS.

Other factors that influence the performance of the different techniques are:

Validation criterion. The choice of validation criterion (see section 2.1.4), which is the measure of performance, can heavily influence the results and thereby the interpretations of what constitutes a successful algorithm. Running the same set of case studies with another criterion may alter the rank between the algorithms.

External parameter values. Different algorithms often require a number of external parameters which have to be adjusted and tuned. Often, a small change in the parameter values can lead to very different results. The higher this number of parameters is, the more time is spent searching for an optimal combination. A parameter set is also domain specific, as different problems require different sets of parameters. When comparing algorithms, finding a suitable set of parameter values should be given the highest priority.

Computation time. Two aspects are involved here, the actual running time of the algorithm and the time spent searching for optimal parameter values. To ensure a comparable level between the algorithms tested, they should all be run and programmed on the same computer. In addition, equal time should be used on all the algorithms when trying to find an optimal set of parameter values. Unfortunately, none of this is gratified in the case studies, since it is mostly referred to the work of others, except when giving the results of the proposed algorithms.

The conclusion is that care should always be taken when doing comparative studies of different algorithms. The results will not necessarily give you the absolute truth about the performance of the techniques, only some ideas of the behavior. Be also aware that the results are only valid for the particular data sets. Further extrapolation and generalization beyond these should be done with extreme care.

5.2 Organization

To simplify comparisons between different data sets, the results are given as normalized root mean square error (NRMSE), which, unlike RMSE, is a relative and not absolute measure of performance defined by

$$\mathrm{NRMSE} = \frac{\sqrt{\sum_{n=1}^{N} (y^n - \hat{y}^n)^2}}{\sqrt{\sum_{n=1}^{N} (y^n - \bar{y})^2}} = \frac{\mathrm{RMSE}}{\sqrt{\frac{1}{N}\sum_{n=1}^{N} (y^n - \bar{y})^2}} \approx \frac{\mathrm{RMSE}}{S(\mathbf{y})} \tag{5.1}$$

If NRMSE is zero the prediction is perfect, whereas a value equal to 100% is equivalent to using the average, $\bar{y}$, as the predictor.

The local modeling algorithms were tested on four different data sets, all of which have previously been used by others when investigating different modeling techniques. The first one is generated from a simulation of a chemical catalytic reactor. The second is obtained from the dynamics of a hydraulic industrial robot. Whereas the last two are taken from the field of NIR spectroscopy, where water content in meat and stiffness in polymers are to be estimated, respectively. In all the data sets there is an anticipated nonlinear relationship between the input and output variables which linear methods might have problems identifying. The data sets are quite different regarding the number of training samples and input variables, the noise level, and the correlation of input variables as shown in table 5.1.

Data set	# of training samples	# of input variables	Noise level	Correlation
Reactor	high (700)	medium (5)	zero	low
Robot	high (800)	low (3)	medium	low
Meat	low (70)	very high (100)	?	high
Polymer	low (47)	very high (138)	?	high

Table 5.1: A rough characterization of the experimental data sets.

Based on the experiences with the artificially generated data sets in chapter 4, the case studies were organized as follows regarding the value of the external parameters K, λ, and A, the type of sample error, and inclusion of the heuristic:

- Of the local PLS modeling algorithms, only **Algorithm 1a** and **1b** were tested.
- All the types of sample error; single, mean, and median were analyzed for both algorithms.
- The number of neighboring samples, K, was not much investigated. Instead, K was given a reasonable value proportional to the size of the training set. This value was then kept fixed after the initial choice, mostly because the computation of the nearest neighboring samples was computational demanding and the time was limited.
- The degree of smoothing between the local models, λ, was given an initial value of $\frac{1}{2}$, and then, more finely tuned both upwards and downwards for the models with initially the lowest minimum NRMSEP.
- The number of latent variables, A_w, in the projected input subspace W was fixed, and the effect of changing it was not investigated. The value was either 3 or 4 depending on the difference in eigenvalue for the principal components.
- The consistency heuristic was included, but only together with a smaller value of K than the initial choice.
- The NRMSEP values shown in the tables are the minimum values. Only the three models with the lowest NRMSEP are displayed.

This process was repeated for all of the four data sets.

Model	Comments	NRMSEP
ASMOD	Quadratic	6%
RBFN	Gaussian	9%
Local PLS	Alg.1a, e1c, $K = 5$, $\lambda = \frac{1}{2}$, 13 local models	10%
Local PLS	Alg.1b, e1c, $K = 5$, $\lambda = \frac{1}{2}$, 8 local models	10%
Local PLS	Alg.1a, e2nc, $K = 20$, $\lambda = \frac{1}{2}$, 13 local models	11%
PLS	Cubic (x), 19 lv	11%
ASMOD	Linear	11%
MLP	5-7-1	13%
PLS	Linear, 5 lv	28%

Table 5.2: Comparison of different models for the simulated chemical reactor.

5.3 Simulated chemical reactor

A catalytic chemical process, transforming unbranched hydrocarbons (nC_5) into branched hydrocarbons (iC_5), is simulated. In the process, hydrogen (H_2) is acting as the catalyst. Two other important variables are the reactor temperature, T, and the flow velocity, V, through the reactor. The chemical reactions between all these variables are described by nonlinear differential equations, based on a real reactor at SINTEF.

The modeling problem is to predict the concentration of iC_5 in the outflow as a function of T, V and $iC5$, $nC5$ and H_2 in the inflow. Data are generated by integrating the equations over different time periods, and with randomly picked initial conditions. The input variables are correlated, since their initial states are not independent of each other. A total of 1000 samples are generated, of which 700 are used in the training set, and the rest are used for testing.

More specific details about the differential equations and the generation of data are given in [Kavli 92].

As seen from the results in table 5.2, the local modeling algorithms performed just as well as the linear ASMOD and the Gaussian RBFN, which perhaps are the two techniques most similar to local PLS, and in fact better than MLP. Generally, all the nonlinear techniques were able to model this large and noiseless data set well, even though there were small individual differences. Except for the local PLS and the linear PLS, all the other results with the different techniques are obtained from [Carlin *et al.* 94], which is referred to for a further discussion.

The values of the external parameters K and A_w were fixed at 20 and 3, respectively, whereas λ was varied and the optimal value was found to be $\frac{1}{2}$. Of the error measures, the single error consistently gave the lowest RMSEP, probably because the data set is noiseless. Generally, the results obtained with the two local modeling algorithms were very similar.

Model	Comments	NRMSEP
ASMOD	Quadratic	15%
LSA	Linear	17%
ASMOD	Linear	17%
RBFN	Gaussian	19%
MLP	3-20-1	23%
PLS	Cubic (x), 10 lv	26%
Local PLS (=MLR)	Alg.1a, e2c, $K = 5$, $\lambda = \frac{1}{2}$, 7 local models	27%
Local PLS (=MLR)	Alg.1a, e2nc, $K = 20$, $\lambda = \frac{1}{4}$, 7 local models	27%
Local PLS (=MLR)	Alg.1b, e3c, $K = 5$, $\lambda = \frac{1}{3}$, 7 local models	27%
PLS (=MLR)	Linear, 3 lv	63%

Table 5.3: Comparison of different models for the hydraulic robot manipulator.

5.4 Hydraulic robot manipulator

The movements of an industrial hydraulic robot manipulator are investigated. Such manipulators have often suffered from the lack of good dynamic models, due to nonlinear hydrodynamic effects involved in the hydraulic components. The goal is to find an empirical model, describing the servo valve control signal, (u), as a function of the joint position (q), velocity ($\dot{q}$), and acceleration ($\ddot{q}$). A more complete description of this experiment can be found in [Kavli 92].

Approximately 40000 samples are generated by sampling corresponding values of u and q, as the manipulator is moving along a randomly generated trajectory. Values of $\dot{q}$ and $\ddot{q}$ are then computed by low pass filtering and numerical differentiations. A linear model is subtracted from the data, leaving mainly nonlinear dependencies. Of the 40000 samples, a training set of 800 and a test set of 200 samples were randomly picked. Note that this training set corresponds to samples 1 to 800, and this test set to samples 801 to 1000 of what is described as the independent test set in other articles [Kavli 92, Johansen and Foss 93]. This reduction was done because of limited time and computer resources. The results obtained with these subsets should still be comparable to those using the large test set and a training set of 8000 samples, because the number of samples was still fairly high and the samples were well distributed in the input space.

The various results with different techniques are given in table 5.3, where the result for the LSA algorithm is taken from [Johansen and Foss 93] and the rest, except those for the local and linear PLS, are obtained from [Carlin *et al.* 94]. For this data set, the local PLS (or rather local MLR since $P = 3$) algorithms were outperformed by the other local methods (LSA, ASMOD, RBFN). One reason is possibly that only 5-7 local models were included in the best local PLS, which is few compared to the 10-13 for the reactor data set. Why the minimum NRMSEP was reached after so few models is uncertain. There should still have been room for improvement since applying the consistency heuristic with $K = 20$ did not change the results very much. Practically all the suggested splitsamples were indeed consistent, indicating large error surfaces having the same sign.

The best local PLS model is plotted in figure 5.1. The graph shows the control signal (u) as a function of joint speed and acceleration, with the joint in the center position i.e. q=0. The best model, with half as large value of λ, is in comparison plotted in figure 5.2. Note how piecewise this model is since the smoothing parameter was very small. The variation of λ did not significantly improve the prediction, as a NRMSEP of at least 30% was always obtained regardless of the degree of smoothing.

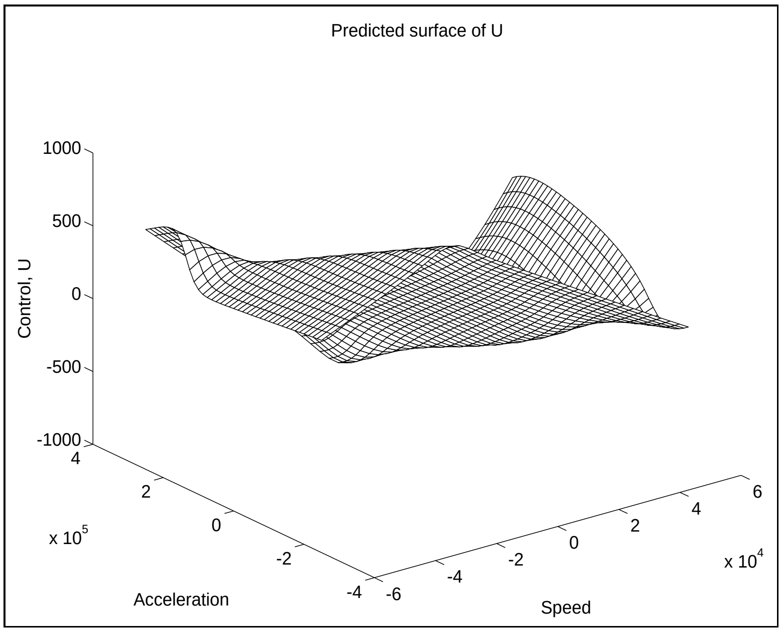

Figure 5.1: Predicted surface of u based on 7 local models, with $\lambda = \frac{1}{2}$ (e2c).

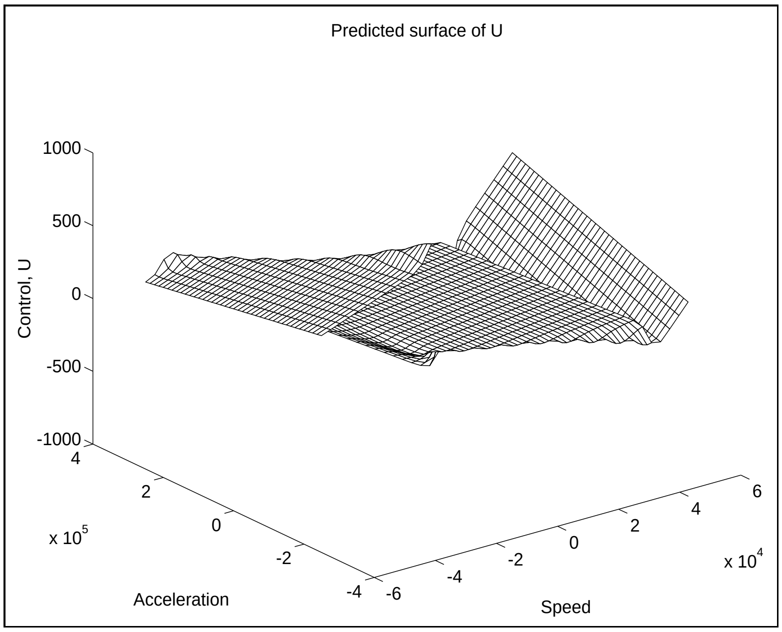

Figure 5.2: Predicted surface of u based on 7 local models, with $\lambda = \frac{1}{4}$ (e2nc).

5.5 NIR spectroscopy

Both of these data sets are taken from the field of spectrometry, where analysis of near infrared diffuse reflectance spectra at different wavelengths, known as the method of NIR spectroscopy, are used to determine different constituents in a substance. Examples are protein, water, or fat content in food products or chemical properties as composition or phase separation in polymers. Different PLS techniques are then used to correlate these spectra to the constituents, since the number of wavelengths is very high and often exceeding the number of samples.

5.5.1 Water estimation in meat

The water concentration in meat is to be predicted based on measurements of NIR transmittance and corresponding percentage of water in 103 beef and pork samples at 100 different wavelengths. Of the 103 samples, 70 are used for training and 33 for testing. The training samples are chosen to span the area of interest as uniformly as possible. Carrying out a linear PCR analysis reveals strong indications of nonlinearity in the data set. All the details are given in [Næs and Isaksson 92].

To reduce multiplicative and additive effects the data are scatter-corrected by the method of Multiplicative Signal Correction (MSC) [Martens *et al.* 83], and only the corrected data are used in the testing.

Model	Comments	NRMSEP
MLP	8 lv, 8-3-1	9.7%
LWR	3 lv, $K = 30$	9.9%
MLP	50-10-1	10.6%
Local PLS	Alg.1a, e2c, $K = 3$, $\lambda = \frac{1}{2}$, 3 local models	11.2%
PCR	Linear, 8 lv	11.7%
Local PLS	Alg.1a, e2nc, $K = 5$, $\lambda = \frac{1}{2}$, 4 local models	12.2%
Local PLS	Alg.1b, e2c, $K = 3$, $\lambda = \frac{1}{2}$, 2 local models	12.4%
PLS	Linear, 6 lv	13.1%

Table 5.4: Comparison of different models for water estimation in meat.

Different results for this data set are given in table 5.4. The results for LWR, PCR, and MLP are obtained from [Næs and Isaksson 92] and [Næs *et al.* 93].

At first sight, the local algorithms seemed to improve the prediction. However, the three results reported in the table were almost the *only* combinations of parameter values that lead to a decrease in the NRMSP. For most of the other combinations, overfitting was the outcome. Even increasing the smoothing between the local models by using a large value of λ, because of high dimensional data, did not help. The conclusion was that the improvements were unreliable, maybe just the result of a lucky division of the training samples and not part of a general tendency.

The reason is probably a combination of many things. At the minimum NRMSEP for linear PLS, the corresponding NRMSEE was over 19%, i.e. much higher than the NRMSEP, indicating that any closer fit to the training samples is very likely to increase the NRMSEP. The small number of training samples made any nonlinear fit difficult, as the number of local models could only be 3–5, before the local PLS algorithms aborted as the result of too few samples in a region. The question is also what really is a significant nonlinear relationship in a 100 dimensional space? Perhaps using this kind of local modeling approach on such an extremely high dimensional problem was an overkill. Applying a nonlinear technique on an almost linear problem can lead to less accurate predictions.

When running the local PLS algorithms, 10 latent variables were the upper limit in each local model. The optimal number was manually selected, by investigating the plot of residual sums of squares from cross-validation. However, virtually no restrictions were put on the number of samples per latent variable. Often a local PLS model was computed using e.g. 10 samples and 4–6 latent variables. The number of principal components in the weighting subspace W was 3.

Note that the linear PCR prediction result was better than that of the linear PLS. A relevant question, which has not been investigated, is whether using PCR as local modeling technique instead of PLS would improve the predictions.

Model	Comments	NRMSEP
MLP	138-25-1	9.3%
Local PLS	Alg.1a, e2c, $K = 3$, $\lambda = \frac{1}{2}$, 2 local models	10.6%
Local PLS	Alg.1b, e2c, $K = 3$, $\lambda = \frac{1}{2}$, 2 local models	10.8%
Local PLS	Alg.1a, e1nc, $K = 5$, $\lambda = \frac{1}{2}$, 3 local models	12.3%
PLS	Linear, 14 lv	13.3%
PCR	Linear, 21 lv	13.5%

Table 5.5: Comparison of different models for stiffness estimation in polymer.

5.5.2 Stiffness estimation in polymer

The goal is to predict the flex modulus (stiffness) of polyurethane elastomers on the basis of the NIR spectra at 138 different wavelengths. The data set contains a total of 90 spectra, of which 47 form the training set and the remaining 43 serve as the test set. The samples in the test set are chosen such that the values of the flex modulus are within the range of the values in the training set. Further details regarding this data set can be found in [Miller and Eichinger 91].

Again, the data are MSC-corrected to reduce multiplicative and additive effects. As was the case with the meat samples, linear PLS analysis shows a possible nonlinear relationship between the input variables and the stiffness [Miller and Eichinger 91].

Various results for this data set are presented in table 5.5. The result with MLP is obtained from [Næs *et al.* 93]. Unfortunately, the same difficulties were present in this NIR data set as in the previous one. Overfitting occurred at once for almost all combinations of parameters in the local PLS algorithms, only 2-3 local models could be computed as the algorithms aborted very quickly, improving the result of the linear PLS was hard etc. The reasons for this behavior are the same as they were for the meat samples. The number of training samples is very small, the number of input variables correspondingly high, possible lack of significant nonlinearity etc.

So, again the apparently better predictions with the local PLS were deceptive, and not a general feature of the algorithms.

One notable difference was the large deviation between the NRMSEE and NRMSEP. For the linear PLS model the NRMSEE was only 6%, i.e. less than half of that of the NRMSEP. This could indicate that even the linear PLS model was initially an overfitted one caused by the few samples and high number of input variables.

Technically, the local algorithms were run with maximum 20 latent variables in each local model. The number of principal components in W was now 4. For this data set applying the consistency heuristic was meaningless unless K was a very small number (3).

5.6 Interpretation of results

When comparing all the experimental results obtained with all the different techniques for the four data sets, the following common features were possible to extract:

- The two local PLS algorithms proposed in this thesis, generally improved the prediction of y compared to linear PLS. This was to be expected since in all the data sets there was an anticipated nonlinear relationship between the input and the output variables. The improvement was very significant for the first two data sets, but less significant for the NIR data sets.
- However, the local PLS algorithms were always outperformed by some other nonlinear technique(s), even though the differences were not much. Thus, the local PLS algorithms were adequate, but not necessarily optimal, ways of modeling nonlinear problems.

Concerning the more specific details when using the two local PLS algorithms:

- None of the two algorithms performed significantly better than the other, even though **Algorithm 1a** tended to work slightly better overall. The reason is that they only differ in how the local validity functions are defined, i.e. how the local models are interpolated. In all other respects are they identical.
- Overfitting was not a problem for the first two data sets. The evolution of the RMSEE and the RMSEP, as the model complexity increased, were very similar. Usually, the RMSEE was a few percent lower than the RMSEP. For the NIR data sets however, overfitting was a major problem. The reason for this discrepancy lies in the different size and dimensionality of the data sets. The first two are large low dimensional data sets, with several hundred training samples. On the other hand, for the NIR data sets the number of input variables is much higher, whereas the number of samples is only a two-digit number. The danger of fitting f too well to the training samples is obviously greater when the samples are few and high dimensional, and all the more so when the already few samples are divided into even smaller disjunct subsets, as is the case in the local PLS algorithms.
- The degree of internally dependent input variables varied from data set to data set. For the reactor the average correlation was 0.19, and for the robot manipulator it was 0.22, as some of the variables were much more correlated than others. On the other hand, the average correlation for the polymer samples was 0.58, and for the meat samples 0.65, i.e. highly correlated variables. Since the local PLS algorithms worked better on the former two data sets, one is tempted to conclude that the algorithms are best suited for problems with moderate degree of internal dependency between the input variables, and not suited for e.g. NIR spectroscopy problems. However, this conclusion is most certainly wrong because the algorithms really did improve the prediction of y when tested on the artificial NIR spectroscopy data set, whose average correlation was 0.81, in the previous chapter. The reason the algorithms did

not perform that well on the real world NIR spectroscopy problems, is more because of few training samples and lack of nonlinearity in the problems than too correlated input variables.

- There were only small differences in the three types of error measure. For one combination of parameters, the single error approach was the best, whereas for another combination on the same data set, the mean or median error gave the lowest RMSEP. The choice of type of sample error is therefore not crucial, even though it should be noted that the combination resulting in the very best prediction usually involved the mean error. The only exception being for the noiseless reactor data set. Not surprising since this type of error measure is less sensitive to outliers than the single error approach is. Why the mean error also worked better than the median error is more difficult to explain.

- The value of the smoothing parameter λ was not crucial either, as the difference in performance for various values was only marginal. One should expect the optimal value to be smaller for **Algorithm 1b** than for **Algorithm 1a**. This was not necessarily the case though. Additionally, the value should probably be larger when the problem was high dimensional, in order to ensure heavier smoothing between the local models. None of this was confirmed by the experiments.

- The inclusion of the consistency heuristic with a small value of K excelled as perhaps the best way of obtaining good predictions with the local PLS algorithms. However, the value of K had to be really small to avoid pre-termination of the algorithms because of no consistent samples.

- Due to time limitations, none of the local PLS algorithms were optimized with regard to K and A_w. Simultaneously doing this for each data set will probably further improve the performance of the algorithms, because it is highly unlikely that the optimal combination of K and A_w was found in the rather ad hoc way these parameter values were selected in the case studies.

Based on these four data sets the conclusion must be that local PLS modeling improves the prediction of y the most when the data set is large and with a distinct nonlinear connection between output and input. When the data set is small and very high dimensional the performance of local PLS modeling is more doubtful.

What is not tested much, is the properties of the local algorithms when the modeling problem is high dimensional, nonlinear, and the number of training samples is fairly large. As indicated by the *artificial* example in section 4.5, this could be a type of problem where local PLS modeling has much to offer. However, such a *real world* data set has not been available during the work of this thesis.

6

Discussion

In this chapter the local modeling algorithms proposed in this thesis are evaluated and their relationship to other nonlinear techniques is discussed. Especially the similarities between RBFN and the local PLS are investigated. Suggestions for further improvement of the algorithms are also presented.

6.1 Evaluation

As all the new algorithms use Gaussian functions for interpolation of the local models, it appears only natural to start by discussing the relationship to some other nonlinear techniques using such functions.

Gaussian RBFN

General RBFN of the form given by equation 2.11 are transformed to

$$f(\mathbf{x}) = \sum_{a=1}^{A} b_a \exp(-\frac{1}{2}(\mathbf{x} - \mu_a)\Lambda^{-1}(\mathbf{x} - \mu_a)^T) \tag{6.1}$$

when the radial basis function h is the Gaussian function ρ, defined in equation 3.3. The parameters that can be estimated by gradient descent are the weights b_a, the centers μ_a, and also the elements of Λ, the diagonal matrix of squared standard deviations. Hierarchical Self-Organized Learning (HSOL) [Lee and Kil 89] and Resource-Allocation Network (RAN) [Platt 91] are examples of learning algorithms, which automatically add new radial basis functions to f in regions where this is necessary during training. The results for RBFN reported in the previous chapter are with Gaussian functions and a modified version of the HSOL algorithm [Carlin 92].

Even though both the model structure in equation 6.1 and the models developed in this thesis (equation 4.2) contain Gaussian functions, there are several differences, as the Gaussian functions are applied very differently. In Gaussian RBFN they are unnormalized and it is the combination of these functions which in fact *is* the modeling surface, and therefore needs fine adjustment, whereas in the proposed algorithms they are only used as normalized smoothers between the local models.

Local model networks

A more closely related network using Gaussian functions as smoothers is the Connectionist Normalized Linear Spline (CNLS) network of [Jones *et. al* 90]. This nonlinear adaptive network has in fact an identical model structure to that of the local PLS algorithms (equation 4.2), i.e.

$$f(\mathbf{x}) = \sum_{m=1}^{M} w_m(\mathbf{t}) f_m(\mathbf{x}) = \sum_{m=1}^{M} \frac{\rho_m(\mathbf{t})}{\sum_{j=1}^{M} \rho_j(\mathbf{t})} \left(\sum_{p=1}^{P} b_{mp} x_p + b_{m0} \right) \tag{6.2}$$

However, the parameters b_{mp} and b_{m0} are estimated by a gradient descent algorithm, and the optimization is global in contrast to local as in the proposed algorithms. The CNLS has only been tested on low dimensional problems, where the number of local models and centers of the ρ_m's are fixed and determined in advance, whereas the standard deviations are trained to produce slightly overlapping functions. In [Jones *et. al* 90] it is suggested that the centers can be trained in a similar manner, by having the Gaussian functions to tend towards those regions of input space where the error is greatest, but no further details are given.

The approach in [Stokbro *et al.* 90] has many of the same features. However, the centers are now determined by an adaptive clustering algorithm, and the standard deviations are computed based on the density of local samples, before all the parameters b_{mp} and b_{m0} are globally optimized by applying a conjugate gradient descent algorithm. When tested on a few problems of predicting the evolution of time series, the approach has yielded good results, even on a medium dimensional example ($P = 6$). The number of training samples has been fairly high, though.

Even more similar is the recently published RBFN approach of [Murray-Smith 94]. Again, the model structure is that of equation 6.2, and the centers of the ρ_m's are again determined by a clustering algorithm. However, the parameters b_{mp} and b_{m0} are then locally optimized for each model m, by using weighted least squares with a diagonal weighting matrix containing the weights $w_m(\mathbf{x}^n)$ corresponding to local sample $\mathbf{x}^n$. Only the N_m local samples belonging to model m are used in the optimization, just as in the local PLS algorithms. The approach has only been tested on the low dimensional robot data set presented in section 5.4, but with good results. The NRMSEP was 19%, which is better than for any of the local PLS algorithms.

Further discussion

The two main novelties in the proposed algorithms are the *iterative* way of decomposing into local models, and the use of *local PLS modeling* in the optimization of the parameter values.

What is new with this iterative approach is the definition of the term splitsample. Other iterative algorithms such as LSA [Johansen and Foss 93], MARS [Friedman 88] and CART [Breiman *et al.* 84] all decompose by examining several different decompositions in each step, and selecting the one which minimizes a global error criterion. In the general **Algorithm 1** of this thesis only one decomposition is carried out. This will probably mean a faster algorithm. The drawback is that the approach is very dependent on finding the *correct* splitsample, in the sense that the splitsample is the position where a new local model has the greatest potential for improving the prediction accuracy. In other words, as the identification of the splitsample is based on estimation errors, it all boils down to defining a measure involving these errors in the best possible way. This is the critical point of all the proposed algorithms. The error measures defined in this thesis are perhaps not the optimal ones.

It can also be argued against the approach that inserting a local model in the region having the anticipated largest deviation (a *max* error), will not necessarily reduce the prediction error (an *expectation* error). Perhaps including a new model elsewhere would have reduced the prediction error more. Selecting an outlier as the splitsample and inserting a local model around that outlier is an illustration of this problem. Thus, such possibilities have tried to be eliminated by introducing the concept of *consistent* errors.

A somewhat similar way of using the estimation errors is applied in the HSOL learning algorithm [Lee and Kil 89] for Gaussian RBFN. A new radial basis function is added to the network when the estimation error of a sample is larger than an error margin, and the sample is not yet covered by a Gaussian function. As in the approach proposed in this thesis, the center of this new Gaussian function is the sample itself.

Applying PLS as the local modeling technique is another feature which distinguishes the proposed algorithms from others. To the best of my knowledge, this has not previously been done. The most comparable is the use of local PCR models in [Næs and Isaksson 91]. The local models are locally optimized. In e.g. the LSA algorithm [Johansen and Foss 93] the optimization of the parameter values is global, whereas in the approach of [Murray-Smith 94] a weighted local optimization is applied. However, both of these approaches use (weighted) *least squares* optimization. When using local *PLS modeling* neither global, nor weighted local optimization appears possible. As an experiment though, **Algorithm 1a** was tested on the low dimensional robot data set (where optimal PLS and MLR are equal), using both global and locally weighted optimization. Global optimization reduced the NRMSEP to 22%, whereas the result for the locally weighted optimization was on the same level as for the local PLS algorithms.

Using prototype vectors (i.e. the splitsamples) and allocating each sample in the input space to the class with the closest prototype vector is not a novel approach, but is frequently used in e.g. data compression, where the method is known as vector quantization [Nasrabadi and King 88]. Much the same way of thinking is also applied in [Næs and Isaksson 91] where each sample is allocated to the class for which it has the largest membership value. The concept of membership value is analogous to using the weights from the validity functions, as is done in this thesis.

What is gained is more *flexible* shaped local regions, as opposed to e.g. the regions in LSA and MARS where the boundary planes are limited to being perpendicular to one of the main axes.

The drawback is that adding a new model will change all the surrounding models, as a number of samples contributing in the computation of these local models are removed and instead used in the formation of the new local model. All the surrounding models will then slightly differ from the old ones in the previous iteration. This is partly avoided in LSA since a new decomposition is restricted to cutting one hyperrectangular box in two pieces i.e. replacing one of the old models by two new. However since the models are globally optimized, more than just that one could be affected.

All the centers, μ_m, of the local validity functions are fixed at the splitsamples, during the whole modeling process in the local PLS algorithms. In other approaches, such as the HSOL learning algorithm, the centers are iteratively updated. One possible way of doing this in the proposed algorithms is adaptive adjustment of the centers, by defining μ_m to be the mean vector of all the input training samples in class m (center of 'mass'), and not the splitsample. The validity functions will then automatically adapt to where the actual data are gathered in a local region.

Another possible improvement is adaptive adjustment of the smoothing parameters, λ_m, as the training progresses. Start with a relatively large value (i.e. much overlap when 'few' models), which is slowly decreased as the number of local models increases (i.e. less overlap when 'many' models). In other words, $\lambda_m^{new} = (1-\varepsilon)\lambda_m^{old}$ in each iteration. This is analogous to decrementing the radius parameter in e.g. the HSOL and the RAN learning algorithms for Gaussian RBFN.

The number of neighboring samples, K, is another external parameter that could be adjusted. One possible way is to replace the parameter by defining two new external parameters K_1 and K_2, where K_1 is the number of neighboring samples used in the computation of the mean and median sample error, and K_2 is the number of consistent samples. The value of K_2 is then typically (much) less than that of K_1. This is advantageous when the data are rather noisy, which, on one hand, requires a small value of K $(= K_2)$ for the consistency heuristic to make any sense without aborting the algorithm, but, on the other hand, a large value of K $(= K_1)$ is preferable for the sample error to yield a good description of the error behavior. The drawback is the inclusion of yet another external parameter to be optimized.

In the proposed algorithms there are three (four) external parameters, K, λ, and A_w, whose values must be specified by the user. In addition, the consistency heuristic and the type of error measure must be selected as well. This is a large number compared to e.g. the ASMOD algorithm, where only the degree of the spline needs to be determined, but not compared to e.g. the HSOL learning algorithm for Gaussian RBFN. In this algorithm, up to nine parameters can be specified by the user. Generally, a large number of external parameters provides a lot of flexibility in the modeling, but also increases the risk of choosing suboptimal parameter values.

One general problem for nonlinear modeling techniques is the dependency on large data sets for training. More samples are needed to obtain reliable models with these techniques than with linear modeling techniques. Unfortunately, this is a problem that has no solution, as the number of samples is limited and collecting new ones is usually either a very time-consuming and expensive process, or simply impossible. Often, the result for local modeling techniques is fewer local models than what is really sufficient to model the system, and less accurate predictions. Local modeling is, therefore, only advisable when the number of training samples is relatively high, compared to the expected number of local models.

6.2 Future work

Future work, with the algorithms proposed in this thesis, could be carried out along the following different lines:

Tests

Additional tests with the present implementation of the algorithms could include:

- How dependent are the prediction results on the values of K and A_w? More time should be spent systematically testing different values for each data set, in order to find the optimal ones, giving an even lower RMSEP.
- K is assumed to be proportional to the number of training samples, N_{train}. Perhaps it will be possible to find a good estimate of the optimal number of K, approximately as a function of N_{train} i.e. $K = h(N_{\text{train}})$?
- Experience with other modeling techniques using a weighting function [Næs and Isaksson 92, Carlin 91] shows that the shape of such a function is not critical. How true is this for the local PLS algorithms?
- It was initially assumed that the proposed algorithms would be sensible to outliers and extremely noisy data sets, because the error measures are based on computing estimation errors. However, no strong indications of such tendencies were observed in the tests. The effect of removing possible outliers were not investigated, though. This should be analyzed more thoroughly.

- Both **Algorithm 2** and **3** (see section 4.6) are anticipated to have better prediction ability than **Algorithm 1**, but will be considerably slower. Exhaustive testing should be carried out in order to validate (or falsify) this hypothesis, and perhaps estimate if the gain is substantial.

Modifications

Additional modifications to the present algorithms, which could be implemented and tested include:

- Replacing the present type of local modeling (linear PLS on the original input variables), by first projecting the input space onto A^* latent variables (with A^* being possibly different from A_w, the dimension of W) found by linear PLS or PCA, and then applying either locally weighted MLR for each local model, as in the approach of [Murray-Smith 94], or a globally optimized weighted least squares algorithm, as in [Johansen and Foss 93], of y on these latent variables. This modification is motivated from the hypothesis that it is possible to improve the predictions, by using nonlinear techniques on the most significant latent variables identified by PLS or PCA. In that respect, the modeling will also be similar to the approach in [Næs and Isaksson 91], but with the additional advantage that weighting of local samples will be possible.

- Adaptive adjustment of λ_m and μ_m, and introduction of the two external parameters, K_1 and K_2, as suggested in section 6.1.

In both these modifications, the iterative decomposition into local models is retained as a fundamental part of the algorithms.

Theoretical justification

The properties of the algorithms should, if possible, be more theoretically explained. Among the aspects in such a statistical foundation are the rate of convergence and estimated accuracy of the modeling technique.

7

Conclusions

In this thesis several new local modeling algorithms, for doing nonlinear empirical modeling, have been proposed. The algorithms were iterative, and based on error analysis of samples, when it came to finding the local models. Linear PLS was applied as the local modeling technique. Each local model was optimized by using only local subsets of samples in the parameter estimation. The local models were interpolated by means of normalized Gaussian weight functions in a principal component subspace, yielding a smooth nonlinear total model. The best of the local PLS algorithms were tested on both low and high dimensional modeling problems. The findings were:

Prediction ability. Under the assumption that there was distinct nonlinearity in the modeling problem, the algorithms provided a smooth total solution f, with more accurate predictions in terms of mean squared error than linear techniques regardless of the dimensionality, P, of the problem.

Novelties. The two main novelties introduced were the iterative decomposition into local models by defining splitsamples, combined with the use of PLS as the local modeling technique. This made the algorithms adaptable to both low and high dimensional problems.

Samples. As with all local modeling algorithms, relatively many training samples were necessary to provide good local models, and thereby a good total solution. The samples should be well-distributed and representative of the system one would like to model.

Parameters. Tuning of the three external parameters, in order to find the optimal values, was computationally demanding. Therefore, the results obtained with the algorithms could probably be improved, by selecting other combinations of parameter values.

Interaction. Determination of the optimal number of latent variables in PLS based on cross-validation had a tendency to result in a too large number of variables. Interactive control during the training process might be useful, although the algorithms could be run without any user-interaction as well.

A

Implementation

The source code was first programmed on a Macintosh IISX, but later transferred to a SGI work station. The programming tool was the mathematical matrix laboratory, MATLAB[a]. Many of the procedures were supported by the "PLS_toolbox" written by Barry M. Wise.

Speed was not one of the important subjects investigated in this thesis. The source code was therefore not programmed very efficiently. Improvements can be made, especially when it comes to computing the K nearest neighboring samples for each training sample. At the moment, this is done using a built-in sort procedure in MATLAB. Perhaps a more efficient algorithm [Fukunaga and Narendra 75] should be used in future implementations.

A few assumptions regarding the PLS algorithm had to be made. The maximum number of PLS factors was set to 10, except in the NIR spectroscopy examples. As suggested in [Martens and Næs 91], at least 4 samples per PLS factor were required. The minimum number of samples in a local region was also set to 4. In addition, the optimal number of PLS factors was found *automatically* using cross-validation (with $V = 10$) on the local samples. The only exception again being in the NIR spectroscopy examples, where the number was *manually* selected.

The complete source code is available on request to the author.

[a]MATLAB is a registered trademark of The Mathworks, Inc.

B

Definitions

B.1 Average correlation

With the average correlation, $\bar{r}$, of a matrix $\mathbf{X}$ is meant the mean value of all the absolute valued inter-variable correlation coefficients. Thus, $\bar{r}$ is a number between 0 and 1, which describes the internal dependencies in the input variables. If $\mathbf{R}$ is the correlation coefficient matrix of $\mathbf{X}$, having size $P \times P$ and with general element $\mathbf{R}_{ij}$, this can be expressed mathematically as

$$\bar{r} = \frac{2}{P(P-1)} \sum_{i<j}^{P} |\mathbf{R}_{ij}| \tag{B.1}$$

with $\frac{P(P-1)}{2}$ being the total number of unique inter-variable coefficients in $\mathbf{R}$.

B.2 Sample statistics

For a realisation, $(x^1, x^2, ..., x^N) = \mathbf{x}$, of size N of a stochastic variable, x, the sample mean, $\bar{x} = \bar{x}(\mathbf{x})$, and sample variance, $S^2 = S^2(\mathbf{x})$, are defined as

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x^i \tag{B.2}$$

and

$$S^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x^i - \bar{x})^2 \tag{B.3}$$

The sample standard deviation, S, of the realisation is defined as $S = S(\mathbf{x}) = \sqrt{(S^2)}$. With a realisation, $(z^1, z^2, ..., z^N) = \mathbf{z}$, of size N of another stochastic variable, z, the sample covariance of the two realisations, $C = C(\mathbf{x}, \mathbf{z})$, is defined as

$$C = \frac{1}{N-1} \sum_{i=1}^{N} (x^i - \bar{x})(z^i - \bar{z}) \tag{B.4}$$

For realisations of A different stochastic variables, $t_1, ..., t_A$, the sample covariance matrix, $\mathbf{P}$, is an $A \times A$ matrix with general element $\mathbf{P}_{ij} = C(\mathbf{t}_i, \mathbf{t}_j)$.

B.3 Distance metrics

When measuring distances between two samples, $\mathbf{t}^i, \mathbf{t}^j \in \mathcal{R}^A$, the distance metric chosen is important.

The Euclidian norm is defined as

$$\|\mathbf{t}^i - \mathbf{t}^j\| = \sqrt{\sum_{a=1}^{A}(t_a^i - t_a^j)^2} = \sqrt{\left(\mathbf{t}^i - \mathbf{t}^j\right)\left(\mathbf{t}^i - \mathbf{t}^j\right)^T} \tag{B.5}$$

and is the natural metric in a Euclidian A-dimensional hyperspace.

The Mahalanobis metric is defined as

$$\|\mathbf{t}^i - \mathbf{t}^j\|_{\mathrm{M}} = \sqrt{(\mathbf{t}^i - \mathbf{t}^j)\mathbf{P}^{-1}\left(\mathbf{t}^i - \mathbf{t}^j\right)^T} \tag{B.6}$$

where $\mathbf{P}$ is the covariance matrix of $\mathbf{T}$. It is reduced to the Euclidian metric if $\mathbf{P}$ equals the identity matrix $\mathbf{I}$.

Bibliography

[Akaike 69] H. Akaike. Fitting autoregressive models for prediction. *Annals. Institute of Statistical Mathematics*, **vol. 21**, pp. 243–247, 1969.

[Bezdec *et al.* 81] J. C. Bezdec, C. Coray, R. W. Gunderson and J. Watson. Detection and Characterization of Cluster Substructure. I. Linear Structure: Fuzzy c-lines. *SIAM Journal on Applied Mathematics*, **vol. 40**, pp. 339–357, 1981.

[Breiman *et al.* 84] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone. *Classification and Regression Trees.* Wadsworths & Brooks, Monterey, California, 1984.

[Carlin 91] M. Carlin. *Neural Nets for Empirical Modeling.* MSc thesis, Norwegian Institute of Technology (NTH), December 1991.

[Carlin 92] M. Carlin. Radial Basis Function Networks and Nonlinear Data Modelling. *Proceedings of Neuro-Nimes'92, Neural Networks and their Applications*, pp. 623–633, 1992.

[Carlin *et al.* 94] M. Carlin, T. Kavli and B. Lillekjendlie. A comparison of four methods for non-linear data modelling. *Chemometrics and Intelligent Laboratory Systems*, **vol. 23**, pp. 163–177, 1994.

[Craven and Wabha 79] P. Craven and G. Wabha. Smoothing noisy data with spline functions. Estimating the correct degree of smoothing by the method of generalized cross-validation. *Numerische Mathematik*, **vol. 31**, pp. 317–403, 1979.

[Dempster *et al.* 77] S. N. Dempster, M. Schatzoff and N. Wermuth. A Simulation Study of Alternatives to Ordinary Least Squares. *Journal of American Statistical Association*, **vol. 72**, pp. 77–91, 1977.

[Farin 88] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press Inc., 1988.

[Folland 84] G. B. Folland. *Real analysis : modern techniques and their applications.* Wiley, New York, 1984.

[Frank 90] I. E. Frank. A Nonlinear PLS Model. *Chemometrics and Intelligent Laboratory Systems*, **vol. 8**, pp. 109–119, 1990.

[Friedman 88] J. H. Friedman. *Multivariate Adaptive Regression Splines.* Technical Report no. 102, Department of Statistics, Stanford University, November 1988.

[Friedman and Stuetzle 81] J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of American Statistical Association*, **vol. 76**, pp. 817–823, 1981.

[Fukunaga and Narendra 75] K. Fukunaga and P. M. Narendra. A Branch and Bound Algorithm for Computing k-Nearest Neighbors. *IEEE Transactions on Computers*, pp. 750–753, July 1975.

[Geladi and Kowalski 86] P. Geladi and B. Kowalski. Partial Least Squares Regression. A tutorial. *Analytica Chimica Acta*, **vol. 185**, pp. 1–17, 1986.

[Gnanadesikan *et al.* 89] Panel on Discriminant Analysis, Classification, and Clustering, Chairman: Ramanathan Gnanadesikan. Discriminant Analysis and Clustering. *Statistical Science*, **vol. 4**, no. 1, pp. 34–69, 1989.

[Hastie and Tibshirani 90] T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models.* Chapman & Hall, London, 1990.

[Hertz *et al.* 91] J. Hertz, A. Krogh and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

[Höskuldsson 88] A. Höskuldsson. PLS Regression Methods. *Journal of Chemometrics*, **vol. 2**, pp. 211–228, 1988.

[Höskuldsson 92a] A. Höskuldsson. The H-principle in modelling with applications to chemometrics. *Chemometrics and Intelligent Laboratory Systems*, **vol. 14**, pp. 139–154, 1992.

[Höskuldsson 92b] A. Höskuldsson. Quadratic PLS Regression. *Journal of Chemometrics*, **vol. 6**, pp. 307–334, 1992.

[Huber 85] P. J. Huber. Projection Pursuit. *The Annals of Statistics*, **vol. 13**, no. 2, pp. 435–475, 1985.

[Johansen and Foss 92a] T. A. Johansen and B. A. Foss. A NARMAX model representation for adaptive control based on local models. *Modeling, Identification, and Control*, **vol. 13**, no. 1, pp. 25–39, 1992.

[Johansen and Foss 92b] T. A. Johansen and B. A. Foss. Nonlinear local model representation for adaptive systems. *International Conference on Intelligent Control and Instrumentation*, **vol. 2**, pp. 677–682, February 1992.

[Johansen and Foss 93] T. A. Johansen and B. A. Foss. *Identification of non-linear System Structure and Parameters using Regime Decomposition.* Technical Report, University of Trondheim, 1993.

[Johnsen and Wichern 88] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis.* Prentice-Hall Inc., New Jersey, 1988.

[Jones *et. al* 90] R. D. Jones *et. al.* Nonlinear Adaptive Networks: A Little Theory, a Few Applications. *Cognitive Modelling in System Control*, Santa Fe, New Mexico, June 10-14, 1990.

[Jordan and Jacobs 94] M. I. Jordan and R. A. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, **vol. 6**, pp. 181–214, 1994.

[Kavli 92] T. Kavli. *Learning Principles in Dynamic Control.* PhD thesis, University of Oslo, September 1993.

[Lee and Kil 89] S. Lee and R. M. Kil. Bidirectional Continuous Associator Based On Gaussian Potential Function Network. *International Joint Conference on Neural Networks*, **vol. 1**, Washington DC, pp. 45–53, 1989.

[Lorber *et al.* 87] A. Lorber, L. E. Wangen and B. R. Kowalski. A Theoretical Foundation for the PLS Algorithm. *Journal of Chemometrics*, **vol. 1**, pp. 19–31, 1987.

[Martens *et al.* 83] H. Martens, S. Å. Jensen, and P. Geladi. *Proceedings, Nordic Symposium on Applied Statistics, Stavanger.* Stokkand Forlag, Stavanger, pp. 205–234, 1983.

[Martens and Næs 91] Harald Martens and Tormod Næs. *Multivariate Calibration.* John Wiley & sons, New York, 1991.

[McClelland *et al.* 86] J. L. McClelland, D. E. Rumelhart, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations and Volume 2: Psychological and Biological Models.* MIT Press, Cambridge, 1986. (only parts)

[Miller and Eichinger 91] C. E. Miller and B. E. Eichinger. Analysis of Reaction-Injection-Molded Polyurethanes by Near-Infrared Diffuse

Reflectance Spectroscopy. *Journal of Applied Polymer Science*, vol. 42, pp. 2169–2190, 1991.

[Moody and Darken 88] J. Moody and C. Darken. Learning with Localized Receptive Fields. *Proceedings of the 1988 Connectionist Models Summer School*, pp. 133–143, 1988.

[Moody and Darken 89] J. Moody and C. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, **vol. 1**, pp. 281–294, 1989.

[Murray-Smith 94] R. Murray-Smith. Local Model Networks and Local Learning. *Fuzzy-Duisburg'94*, University of Duisburg, April 7.-8., 1994.

[Nasrabadi and King 88] N. M. Nasrabadi and R. A. King. Image Coding Using Vector Quantization. *IEEE Transactions in Communications*, **vol. 36**, pp. 957–971, 1988.

[Næs 91] T. Næs. Multivariate Calibration when Data are Split into Subsets. *Journal of Chemometrics*, **vol. 5**, pp. 487–501, 1991.

[Næs *et al.* 90] T. Næs, T. Isaksson and B. Kowalski. Locally Weighted Regression and Scatter Correction for Near-Infrared Reflectance Data. *Analytical Chemistry*, **vol. 62**, no. 7, pp. 664–673, 1990.

[Næs *et al.* 93] T. Næs, K. Kvaal, T. Isaksson and C. Miller. Artificial Neural Networks in Multivariate Calibration. *Journal of Near Infrared Spectroscopy*, **vol. 1**, pp. 1–11, 1993.

[Næs and Isaksson 91] T. Næs and T. Isaksson. Splitting of Calibration Data by Cluster Analysis. *Journal of Chemometrics*, **vol. 5**, pp. 49–65, 1991.

[Næs and Isaksson 92] T. Næs and T. Isaksson. Locally Weighted Regression in Diffuse Near-Infrared Transmittance Spectroscopy. *Applied Spectroscopy*, **vol. 46**, no. 1, pp. 34–43, 1992.

[Platt 91] J. Platt. A Resource Allocating Network for Function Interpolation. *Neural Computation*, **vol. 3**, no. 2, pp. 213–225, 1991.

[Poggio and Girosi 90] T. Poggio and F. Girosi. Networks for Approximation and Learning. *Proceedings of the IEEE*, **vol. 78**, no. 9, pp. 1481–1497, 1990.

[Rumelhart *et al.* 86] D. E. Rumelhart, G. E. Hinton and R. J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, **vol. 323**, pp. 533–536, 1986.

[Seasholtz and Kowalski 93] M. B. Seasholtz and B. Kowalski. The Parsimony Principle Applied to Multivariate Calibration. *Analytica Chimica Acta*, **vol. 277**, no. 2, pp. 165–177, 1993.

[Stokbro *et al.* 90] K. Stokbro, D. K. Umberger and J. A. Hertz. Exploiting Neurons with Localized Receptive Fields to Learn Chaos. *Complex Systems*, **vol. 4**, pp. 603–622, 1990.

[Stone 74] M. Stone. Cross-validatory Choice and Assessment of Statistical Predictions. *Journal of Royal Statistical Society B*, **vol. 36**, pp. 111–133, 1974.

[Stone and Brooks 90] M. Stone and R. J. Brooks. Continuum Regression: Cross-validated Sequentially Constructed Prediction Embracing Ordinary Least Squares, Partial Least Squares and Principal Component Regression. *Journal of Royal Statistical Society B*, **vol. 52**, no. 2, pp. 237–269, 1990.

[Wahba 90] G. Wahba. Spline Models for Observational Data. *Society for Industrial and Applied Mathematics*, Philadelphia, Pennsylvania, 1990.

[Wold 66] H. Wold. Nonlinear Estimation by Iteratives Least Squares Procedures. *Research Papers in Statistics*, F.David (editor), Wiley, New York, pp. 411–444, 1966.

[Wold *et al.* 84] S. Wold, A. Ruhe, H. Wold and W. J. Dunn III. The collinearity problem in linear regression. The PLS approach to generalized inverses. *SIAM Journal of Science and Statistical Computation*, **vol. 5**, pp. 734–743, 1984.

[Wold *et al.* 87] S. Wold, K. Esbensen and P. Geladi. Principal Component Analysis. A Tutorial. *Chemometrics and Intelligent Laboratory Systems*, **vol. 2**, pp. 37–52, 1987.

[Wold *et al.* 89] S. Wold, N. Kettaneh-Wold and B. Skagerberg. Nonlinear PLS Modeling. *Chemometrics and Intelligent Laboratory Systems*, **vol. 7**, pp. 53–65, 1989.

[Wold 92] S. Wold. Nonlinear Partial Least Squares Modelling II.Spline Inner Relation. *Chemometrics and Intelligent Laboratory Systems*, **vol. 14**, pp. 71–84, 1992.