# Regression Estimation with Support Vector Learning Machines

Alexander Smola[1]

in collaboration with

Chris Burges[2]
Harris Drucker[3]
Steve Golowich[4]
Leo van Hemmen[5]
Klaus-Robert Müller[6]
Bernhard Schölkopf[7]
Vladimir Vapnik[8]

December 31, 1996
Version 1.01

[1]Physik Department, Technische Universität München
[2]Lucent Technologies, Holmdel
[3]Lucent Technologies, Holmdel
[4]Princeton University, Princeton
[5]Physik Department, Technische Universität München
[6]GMD First, Berlin
[7]Max Planck Institut für biologische Kybernetik
[8]AT&T Research, Holmdel

Support Vector Learning Machines (SVLM) have become an emerging technique which has proven successful in many traditionally neural network dominated applications. This is also the case for Regression Estimation (RE). In particular we are able to construct spline approximations of given data independently from the number of input-dimensions regarding complexity during training and with only linear complexity for reconstruction - compared to exponential complexity in conventional methods.

I would like to thank Yann LeCun, Patrick Haffner, Martin Hasler, Darshyang Lee, Noboru Murata, Craig Nohl, Patrice Simard, Charles Stenard and Pascal Vincent for many inspiring discussions. This project was made possible through funding by ARPA and the GNSF.

*This text was created with $\LaTeX\,2_\varepsilon$ using the document class article and the german package. It can be downloaded from http:/www.first.gmd.de/˜smola*

# Contents

# List of Figures

# Chapter 0

# A Roadmap

The purpose of this text is threefold: First it should be a documentation of the research done at Bell Laboratories, Holmdel. Secondly it should serve as an introduction to Support Vector Regression-Estimation and finally it is written in order to obtain the degree of a “Diplomingenieur der Physik” at the Technische Universität München. Of course these goals may sometimes not be consistent. Hence the reader may skip some chapters and focus on other ones to get the information he needs. I tried to make the chapters as self contained as possible without repeating tedious details.

## 0.1 How to read this Thesis

Chapter 1 contains a short overview over some basic regression techniques and explains the fundamental ideas of linear Support Vector function approximation including basic knowledge about convex programming necessary for solving the optimization equations involved. The reader may skip the parts not directly concerning Support Vectors as these are just for giving a motivation for the setting of the problem.

In chapter 2 I will show how Support Vector Machines can be extended to the case of Regression/Estimation with noisy data. A set of different loss functions and their implications on the robustness of the estimation will be presented with an explicit derivation of the corresponding optimization functionals. For a brief overview it may suffice to read the section about $\epsilon$-insensitive loss functions only.

The next chapter describes a way of porting the ideas on Linear Support Vector Regression to nonlinear models. The approach is similar to nonlinear Support Vector pattern recognition following the concept of [ABR64] and [BGV92] of mapping to a highdimensional space where linear calculations are performed. For a reader already familiar with this concept and for a quick review of the methods employed the sections 3.4 and 3.5 may be enough. A general method of constructing nonlinear regression systems is given in 3.6.

Chapter 4 describes the more implementation specific solutions of the support vector system. Basically any quadratic programming algorithm could be used for solving the problem although different methods may show varying performance due to the setting of the problem. An exemplary solution is shown for interior point methods following the ideas of [Van94]. Furthermore this chapter contains a method for efficiently solving Toeplitz-like systems by a modified Levinson method.

Experimental results on the performance of Support Vector Regression are given in chapter 5. Further information may be obtained from [VGS96] and [DBK$^+$96].

Finally appendix A concludes this work by describing some of the implementation issues of Support Vector learning machines. This may be relevant for somebody trying to model a similar system only. Hence most of the readers will want to skip it. Appendix B contains a brief definition of the industry standard MPS file format.

## 0.2 A Short Review of Approximation and Regression Estimation

The simplest way of approximating a function (and the crudest one, too) would be to approximate it by its mean in the domain of interest. A more sophisticated approach would be to choose linear functions or more complicated bases of functions to achieve this goal. Increasing the complexity of the base seems to be the solution to obtain better results. But this is not really true as one will encounter the well known effect of 'over–fitting', which means that the complexity of the system of functions used is too high.

Since Schoenberg's pioneering paper [Sch46] on spline approximation consisting of $l$-times differentiable piecewise polynomial functions combined with a set of smoothness (and regularity) constraints have been investigated in great detail [PTVF92], [SB93]. This approach has proven to be effective in the case of low-dimensional functions only as the number of coefficients increases exponentionally with dimensionality thereby virtually imposing a limit of four dimensions in real world applications. Decompositions into orthogonal systems [Tim60], [Wal94] like Legendre–, Laguerre–, Chebycheff–, Hermite–, Haar–bases and the family of Fourier Transforms, Windowed Fourier Transforms, Wavelets with tight frames or orthogonal systems [Dau92] were investigated in order to achieve efficient ways for true and/or fast function approximation and data compression.

All of the approaches described above (except approximation by nonseparable wavelet bases) share the problem of exponentional increase in the number of coefficients with the dimensionality of the problem. A solution was the way to nonseparable expansions, e.g. ANOVA–decompositions [SHKT95], Neural Networks and other methods like CART [BFOS84], MARS [Fri91] ... These methods stem from regression estimation and allow tractable solutions of high dimensional problems. This brings us to the crucial question: Why do we need another method for function approximation or regression if there's already such a large variety of concepts available that should suit everyone's need?

## 0.3 The Reason for Support Vectors

The reason is that all of these methods have some shortcomings which we tried to overcome in this approach. Neural Networks which are probably the most popular way of doing high dimensional regression estimation have two drawbacks. Their architecture has to be determined a priori or modified while training by some heuristic which results in a non necessarily optimal structure of the network and can become a difficult combinatorial problem for multilayer networks. Still they are universal approximators [Cyb88], [Mur96b].

Secondly ways of regularization (besides the selection of the architecture) are rather limited. They consist of early stopping [DH74], training with noise [Bis95b], weight decay [SL92a] and similar methods. Unfortunately Neural Networks can get stuck in local minima while training. For some methods of training (early stopping) it is not even desired for the network to obtain its minimum. Hence a part of the capacity of the network is (sometimes deliberately) wasted in order to obtain better generalization performance. Howewer this gives only a very indirect way of controlling the capacity of the system and is far from being optimal.

Only for the asymptotic case (which in reality rarely occurs, still there are some results hinting that speech recognition might be such a case) [MYA94], [Aka74] and for the case of known prior probabilities (which is not a realistic assumtion) optimal selection criteria have been obtained. Only recently good bounds on the complexity of Neural Networks have been computed by McIntyre and Karpinski [KM95] and Hole [Hol96]. Still the problem of effectively controlling the capacity after having selected the architecture remains. Moreover after training it is rather hard to find a meaningful interpretation of the weights.

CART conversely uses a purity functional for pruning instead of more directly motivated regularization techniques. It also neglects the loss function specific to the problem to be solved. Moreover it is quite restrictive in terms of the admissible models. Instead it would be desirable to have a great liberty in choosing the type of functions one would like to approximate the data with and thereby adding prior knowledge to the way the approximation is constructed.

But instead of giving a definition by exclusion the advantages of Support Vector Regression Estimation can be summed up easily. The architecture of the system doesn't have to be determined before training. Input data of any arbitrary dimensionality can be treated with only linear cost in the number of input dimensions. A unique solution is found after training as solution of a quadratic programing problem. The modeling functions may be chosen among a great variety having to satisfy only some condition from Hilbert-Schmidt theory. Capacity can be controlled effectively through the regularization functional used (some more research will have to be done on this point) by the same method that was applied to Support Vector Pattern Recognition [GBV93].

# Chapter 1

# Introduction

Before any calculus is done one should consider the basic problem we are dealing with in its fundamental setting: to approximate a function from given data and/or to estimate a function from given data. There is an important difference between these two problems. In the first case we know that our data is correct, viz. our measurements were taken without noise and we would like to obtain a function that assumes the same values (up to a given precision) as our measurements did. Hence this is a setting closely related to the standard interpolation problems.

Conversely in the problem of function estimation we can not trust our data completely as it may have been generated with some additional noise. In most of the cases we do not even know the noise model but only have some clues what it might be alike. Still both of these two problems can be treated with just one formalism, the distinction between them being the choice of an appropriate cost function (cf. chapter 2).

## 1.1 The Regression Problem

Suppose we are given a set of observations generated from an unknown probability distribution $P(\vec{\mathbf{x}}, y)$[1]

$$X = \{(\vec{\mathbf{x}}_1, y_1), (\vec{\mathbf{x}}_2, y_2), \dots \; (\vec{\mathbf{x}}_l, y_l)\} \text{ with } \vec{\mathbf{x}}_i \in \mathbb{R}^n, y_i \in \mathbb{R} \tag{1.1}$$

and a class of functions

$$F = \{f | f : \mathbb{R}^n \longmapsto \mathbb{R}\} \tag{1.2}$$

then the basic problem is to find a function $f \in F$ that minimizes a risk functional

$$R[f] = \int l(y - f(\vec{\mathbf{x}}), \vec{\mathbf{x}}) \; dP(\vec{\mathbf{x}}, y) \tag{1.3}$$

[1]Sometimes the sampling points are fixed and $P(x)$ is known, too, e.g. audio data.

$l$ is a loss function, i.e. it indicates how differences between $y$ and $f(\vec{\mathbf{x}})$ should be penalized. A common choice is $l(\eta, \vec{\mathbf{x}}) = |\eta|^p$ for some positive number $p$ (for interpolation one could require $l(\eta, \vec{\mathbf{x}}) = \begin{cases} 0 & \text{for} \quad \eta = 0 \\ \infty & \text{for} \quad \eta \neq 0 \end{cases}$.

As $P(\vec{\mathbf{x}}, y)$ is unknown one cannot evaluate $R[f]$ directly but only compute the empirical risk

$$R_{emp} = \frac{1}{l} \sum_{i=1}^{l} l(y_i - f(\vec{\mathbf{x}}_i), \vec{\mathbf{x}}_i) \tag{1.4}$$

and try to bound the risk $R$ by $R_{emp} + R_{gen}$ where $R_{gen}$ is an upper bound of the generalization error [Vap79] depending on the class of functions used to approximate the data given.

## 1.2 A Special Class of Functions

A simple choice of $F$ is

$$F = \operatorname{span}\{\phi_1(\vec{\mathbf{x}}), \phi_2(\vec{\mathbf{x}}), \ldots\ \phi_n(\vec{\mathbf{x}})\} \text{ with } \phi_j(\vec{\mathbf{x}}) : \mathbb{R}^n \longmapsto \mathbb{R} \tag{1.5}$$

Here $F$ is the space spanned by the linear combination of a set of functions. For simplicity we will assume that these are linearly independent hence constitute a base of $F$. Then the basic regression problem consists of finding a set of scalars

$$A = \{\alpha_1, \alpha_2, \ldots\ \alpha_n\} \text{ with } f(\vec{\mathbf{x}}) = \sum_{j=1}^{n} \alpha_j \phi_j(\vec{\mathbf{x}}) \tag{1.6}$$

such that $R[f] = R[\vec{\alpha}]$ is minimized.

### 1.2.1 Least Mean Square Fitting

As described above $R[f]$ is unknown. A very naive approach would be to minimize $R_{emp}$ only with $l(\eta) = \eta^2$ or in other words to minimize the $L_2$–norm[2].
Denote

$$\begin{array}{rcl} \Phi_{ij} & = & \phi_j(\vec{\mathbf{x}}_i) \\ \text{and } \vec{\mathbf{y}} & = & (y_1, y_2, \ldots\ y_l) \end{array} \tag{1.7}$$

then

$$\begin{array}{rcl} R_{emp} & = & \sum_{i=1}^{l} \left( \sum_{j=1}^{n} \phi_j(\vec{\mathbf{x}}_i) \alpha_j - y_i \right)^2 \\ & = & \|\Phi\vec{\alpha} - \vec{\mathbf{y}}\|_2^2 \end{array} \tag{1.8}$$

[2] $||.||_p$ denotes the $p$–norm, hence $||\vec{\mathbf{x}}||_p = (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$ for $\vec{\mathbf{x}} \in \mathbb{R}^n$ and $||\Psi(x)||_p = (\int |\Psi(x)|^p dx)^{\frac{1}{p}}$ for $\Psi \in L_p$

The minimum of the quadratic function $R_{emp}$ is obtained for $\vec{\alpha}^{min}_{LMS}$ with

$$\begin{array}{lrcl} & \Phi^t\Phi\,\vec{\alpha}^{min}_{LMS} - \Phi^t\vec{\mathbf{y}} & = & 0 \\ \Longleftrightarrow & \vec{\alpha}^{min}_{LMS} & = & \left(\Phi^t\Phi\right)^{-1}\Phi^t\vec{\mathbf{y}} \end{array} \tag{1.9}$$

In the eigensystem of $\Phi^t\Phi$, $\{\lambda_i, \vec{\mathbf{a}}_i\}$ we get

$$\begin{array}{lrcl} & \vec{\alpha}^{min}_{LMS} & = & \sum_i k_i\vec{\mathbf{a}}_i \\ \Longrightarrow & \Phi^t\Phi\,\vec{\alpha}^{min}_{LMS} & = & \sum_i \lambda_i k_i\vec{\mathbf{a}}_i = \Phi^t\vec{\mathbf{y}} \end{array} \tag{1.10}$$

This method does not contain any means of capacity control (besides choosing a smaller set of functions) which makes it very sensitive to overfitting and noisy data. Epecially for the case of $l < n$ when we have more functions at hand than measurements the problem is ill posed as $\operatorname{rank}\left(\Phi^t\Phi\right) \leq l$. Therefore $\left(\Phi^t\Phi\right)^{-1}$ is not well defined in this case. One may add a term of the form $\|\vec{\alpha}\|_2^2$ or use a Singular Value Decomposition. This constitutes a regularizer in the sense of Arsenin and Tikhonow [TA77] and brings us to the case of Ridge Regression.

## 1.2.2 Ridge Regression

$$R_{ridge} = R_{emp} + \gamma\|\vec{\alpha}\|_2^2 \tag{1.11}$$

For the case of the Least Mean Square Fit ($l(\eta) = \eta^2$) this yields

$$\begin{array}{rcl} R_{ridge} & = & \|\Phi\vec{\alpha} - \vec{\mathbf{y}}\|_2^2 + \gamma\|\vec{\alpha}\|_2^2 \\ & = & \vec{\alpha}^t\left(\Phi^t\Phi + \gamma\mathbf{1}\right)\vec{\alpha} - 2\vec{\mathbf{y}}^t\Phi\vec{\alpha} + \vec{\mathbf{y}}^t\vec{\mathbf{y}} \end{array} \tag{1.12}$$

As in (1.8) we solve for $\vec{\alpha}^{min}_{ridge}$ and express them in terms of $\vec{\alpha}^{min}_{LMS}$ in the eigensystem of $\Phi^t\Phi$, $(\lambda_i, \vec{\mathbf{a}}_i)$. So we get

$$\vec{\alpha}^{min}_{ridge} = \sum_i \frac{\lambda_i}{\lambda_i + \gamma} k_i\vec{\mathbf{a}}_i \tag{1.13}$$

**Proof:**

For the minimum we need

$$\begin{array}{rcl} \frac{\partial R_{ridge}}{\partial\vec{\alpha}} & = & 2\left(\Phi^t\Phi + \gamma\mathbf{1}\right)\vec{\alpha} - 2\Phi^t\vec{\mathbf{y}} \\ & = & 0 \end{array} \tag{1.14}$$

Substituting $\vec{\alpha}^{ridge}_{min}$ into $\frac{\partial R_{ridge}}{\partial\vec{\alpha}}$ we get

$$\begin{array}{rcl} \frac{\partial R_{ridge}}{\partial\vec{\alpha}} & = & 2\sum_i\left((\lambda_i + \gamma)\left(k_i\frac{\lambda_i}{\lambda_i+\gamma}\right)\vec{\mathbf{a}}_i\right) - 2\Phi^t\vec{\mathbf{y}} \\ & = & 2\left(\sum_i\left(\lambda_i k_i\vec{\mathbf{a}}_i\right) - \Phi^t\vec{\mathbf{y}}\right) \\ & = & 0 \end{array} \tag{1.15}$$

□

The term $\sum_i \frac{\lambda_i}{\lambda_i+\gamma}$ is often called the effective number of free parameters (cfr. [Bis95a]) and used for a Bayesian regularization argument.

## 1.3 The $\epsilon$-Precision Approximation Problem: Linear Support Vector Regression

Now let us consider a very simple cost function

$$l(\eta)_\epsilon = \begin{cases} \infty & \text{if } \eta \; < \; -\epsilon \\ 0 & \text{otherwise} \\ \infty & \text{if } \eta \; > \; \epsilon \end{cases} \tag{1.16}$$

This cost function stems from Support Vector Pattern Recognition [BGV92], [CV95] where a similar function $l_{SVPR}(\eta, \vec{\mathbf{x}})$ had been used

$$l_{SVPR}(\eta, \vec{\mathbf{x}}) = \begin{cases} 0 & \text{if } \eta \; > \; 1 \\ \infty & \text{otherwise} \end{cases} . \tag{1.17}$$

In this case $\eta$ is the output of a binary classifier (for a positive sample - the signs change accordingly for negative samples). The modification for regression estimation lies in the fact that we get an upper and a lower constraint on the output of the system.

Secondly let us consider a simple class of functions $F$ namely the set of linear functions.

$$F = \{ f \mid f(\vec{\mathbf{x}}) = (\vec{\omega}, \vec{\mathbf{x}}) + b, \; \vec{\omega} \in \mathbb{R}^n \} \tag{1.18}$$

We would like to find the function with the smallest steepness among the functions that minimize the empirical risk functional.

$$R_{SV} = R_{emp} + \gamma \|\vec{\omega}\|_2^2 \tag{1.19}$$

The functional $R_{SV}$ minimizes the empirical risk functional for (1.16) indeed as $R_{emp}$ only may assume the values 0 and $\infty$. So effectively its solution is the flattest linear function for which $R_{emp} = 0$.

In general finding the "flattest" linear function corresponds to minimizing $\|\vec{\omega}\|_p$ with some suitable $p \in \mathbb{R}^+$. For $p = 2$ this optimization problem becomes a quadratic one which is much more tractable with standard optimization techniques. Also from the equivalence of all norms in $\mathbb{R}^n$ follows that an optimal solution for $p_0$ will be a good solution for $\tilde{p}$ as well. Hence for the rest of this text we will assume $p = 2$.

In the case of a loss function of type (1.16) the functional (1.19) gives the following quadratic optimization problem.

$$\begin{array}{rl} \text{minimize} & L(\vec{\omega}, b) = \frac{1}{2}\|\vec{\omega}\|_2^2 \\ & \\ \text{subject to} & \varepsilon - (f(\vec{\mathbf{x}}_i) - y_i) \geq 0 \quad \forall \, i \in \; [1..l] \\ & \varepsilon - (y_i - f(\vec{\mathbf{x}}_i)) \geq 0 \\ \text{where} & f(\vec{\mathbf{x}}) = \vec{\omega}^t \vec{\mathbf{x}} + b \end{array} \tag{1.20}$$

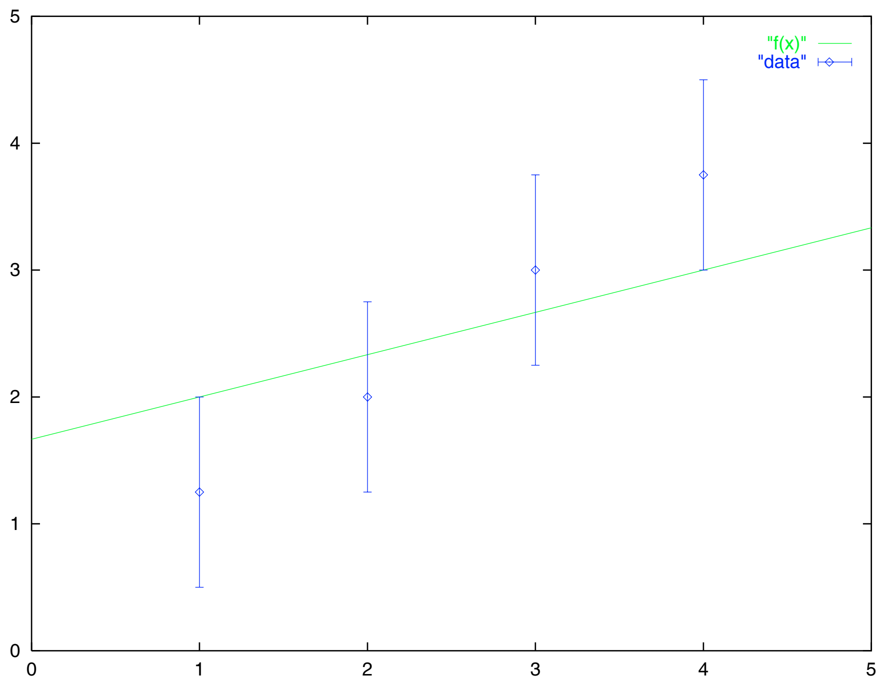


Figure 1.1: sample data and corresponding flattest fit

**Example 1 (Hard Margin Approximation)** *Figure 1.1 shows an approximation of some datapoints with fixed precision (indicated by the errorbars). Among the admissible functions that would satisfy these constraints the flattest one is selected.*

In order to solve (1.20) we need some basic knowledge from optimization theory which will be presented here without great detail. For the reader who is interested in getting a greater insight into these problems any book on operations research and mathematical programming will suit his needs, e.g. [Fle89].

## 1.4 The Lagrange Function and Wolfe's Dual

The basic setting of convex programming is the following problem.
**Optimization Problem:**

$$\begin{array}{lrcll} \text{minimize} & f(\vec{\mathbf{x}}) & & & \\ \text{subject to} & c_i(\vec{\mathbf{x}}) & \geq & 0 & \text{for } i \in [1..m_{ineq}] \\ \text{subject to} & c_i(\vec{\mathbf{x}}) & = & 0 & \text{for } i \in [m_{ineq}+1..m] \end{array} \tag{1.21}$$

Both $f(\vec{\mathbf{x}})$ and $c_i(\vec{\mathbf{x}})$ are required to be convex functions. Hence the constraints $c_i(\vec{\mathbf{x}})$ define a convex domain, the primal feasible region. It can be shown that the system (1.21) has a unique solution if $f$ is strictly convex and the equality constraints are linear (or in other words both the equality constraints $c_i(\vec{\mathbf{x}})$ and $-c_i(\vec{\mathbf{x}})$ are convex for $in \in [m_{ineq}+1..m]$).

**Proof:**

By contradiction: Suppose we have two points $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$ being an optimal solution, then obviously

$$f(\mu\vec{\mathbf{x}}_1 + (1-\mu)\vec{\mathbf{x}}_2) < \mu(f(\vec{\mathbf{x}}_1) + (1-\mu)f(\vec{\mathbf{x}}_2) \tag{1.22}$$

but by the convexity of the feasible region $\mu\vec{\mathbf{x}}_1 + (1-\mu)\vec{\mathbf{x}}_2$ is feasible, too, hence $\vec{\mathbf{x}}_1$ and $\vec{\mathbf{x}}_2$ can't be optimal. $\square$

Analogously to the well known methods in classical mechanics (cfr. [Gol86]) we define the Lagrange function

$$L(\vec{\mathbf{x}}, \lambda_1, \ldots \lambda_m) = f(\vec{\mathbf{x}}) - \sum_{i=1}^{m} c_i(\vec{\mathbf{x}})\lambda_i \tag{1.23}$$

where $\lambda_i$ are the so called Lagrange multipliers, also called dual variables. The feasible region for $\lambda_i$ is $\mathbb{R}_0^+$ for $i \in [0..m_{ineq}]$ and $\mathbb{R}$ for $i \in [m_{ineq}+1..m]$

Now we want to obtain the dual problem (Wolfe's Dual) from $L$ namely the optimization problem in the dual variables $\vec{\lambda}$ which results to be a maximization problem.

Note that iff $L(\vec{\mathbf{x}}, \vec{\lambda})$ has a saddle–point $(\vec{\mathbf{x}}_0, \vec{\lambda}_0)$ in the admissible domain $\mathbb{R}^n \otimes \mathbb{R}_0^{+m_{ineq}} \otimes \mathbb{R}^{m-m_{ineq}}$ then this point is the global optimum of 1.21. A saddle–point is characterized by

$$L(\vec{\mathbf{x}}, \vec{\lambda}_0) \geq L(\vec{\mathbf{x}}_0, \vec{\lambda}_0) \geq L(\vec{\mathbf{x}}_0, \vec{\lambda}) \tag{1.24}$$

for all $\vec{\mathbf{x}}$ and $\vec{\lambda}$. Hence in order to compute $(\vec{\mathbf{x}}_0, \vec{\lambda}_0)$ we will solve $L$ for $\vec{\mathbf{x}}$.

$$\frac{\partial L(\vec{\mathbf{x}}, \vec{\lambda})}{\partial \vec{\mathbf{x}}} = 0 \tag{1.25}$$

This yields the corresponding dual problem in $\vec{\lambda}$:

$$g(\vec{\lambda}) := L(\vec{\mathbf{x}}(\vec{\lambda}), \vec{\lambda}) \tag{1.26}$$

$$\begin{array}{lllll} \text{maximize} & g(\vec{\lambda}) & & & \\ \text{subject to} & \lambda_i & \geq & 0 & \text{for } i \in [1..m_{ineq}] \\ & \lambda_i & \in & \mathbb{R} & \text{for } i \in [m_{ineq}+1..m] \end{array} \tag{1.27}$$

The saddle–point conditions also can be written as follows

$$\begin{array}{rl} & f(\vec{\mathbf{x}}) = g(\vec{\lambda}) \\ \text{and} & (\vec{\mathbf{x}}, \vec{\lambda}) \text{ feasible} \end{array} \tag{1.28}$$

We will use this equality in chapter 4 for determining the duality gap between the primal and dual set of variables in order to measure the convergence of an interior point algorithm.

## 1.5 Karush-Kuhn-Tucker theorem

From (1.24) it can be seen that

$$\lambda_i c_i(\vec{\mathbf{x}}) = 0 \;\; \forall\, i \;\in\; [1..m] \tag{1.29}$$

In other words this means that only active constraints may result in Lagrange multipliers not equal to zero which is a well known fact from classical mechanics. There the multipliers represent the virtual forces resulting from the constraints of the problem. This is of great importance for the solution of the Support Vector problem as it allows a significant simplification of the regression - only the datapoints with nonvanishing Lagrange multipliers have to be taken into account.

## 1.6 The Dual Support Vector Problem

Solving (1.20) with the methods described above we get (for the future we will use $\vec{\alpha}$, $\vec{\alpha}^*$ instead of $\vec{\lambda}$ in order to be consistent with the existing literature on Support Vectors)

$$\begin{aligned} L(\vec{\omega}, b, \vec{\alpha}, \vec{\alpha}^*) = \;& R_{SV}(\vec{\omega}, b) \\ & - \textstyle\sum_{i=1}^{l} \alpha_i(\varepsilon - (y_i - f(\vec{\mathbf{x}}_i)) \\ & - \textstyle\sum_{i=1}^{l} \alpha_i^*(\varepsilon - (f(\vec{\mathbf{x}}_i) - y_i))). \end{aligned} \tag{1.30}$$

Now we compute the partial derivatives of $L$ wrt. the primal variables

$$\frac{\partial L}{\partial \vec{\omega}} = \vec{\omega} - \sum_{i=1}^{l} \alpha_i \vec{\mathbf{x}}_i + \sum_{i=1}^{l} \alpha_i^* \vec{\mathbf{x}}_i \tag{1.31}$$

$$= 0$$

$$\Longrightarrow \quad \vec{\omega} = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)\, \vec{\mathbf{x}}_i \tag{1.32}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \tag{1.33}$$

$$\Longrightarrow \quad 0 = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \tag{1.34}$$

substituting (1.32) and (1.34) back into (1.30) yields:

$$\begin{aligned} \text{maximize } g(\vec{\alpha}, \vec{\alpha}^*) \;&=\; -\frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) \\ & \quad + \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) y_i \\ \text{with} \quad & \vec{\alpha}, \vec{\alpha}^* \;\in\; \mathbb{R}^n \end{aligned} \tag{1.35}$$

Unfortunately this model breaks down when we encounter a non-realizable case (or data with noise) as the primal feasible region becomes zero. This means that there would be no function that could approximate the given data with $\epsilon-$precision. For instance consider the pathological case of $\vec{\mathbf{x}}_i = \vec{\mathbf{x}}_{\hat{i}}$ and $|y_i - y_{\hat{i}}| > 2\epsilon$.

Additionally it does not provide any means for capacity control. A way to cope with both issues is to modify the loss function $l(\eta)$. This will be done in the next chapter.

# Chapter 2

# Loss Functions

For the case of data with noise we will have to consider replacing the loss function (1.16) with another one that doesn't have singularities. Moreover it is required to be convex in order not to spoil the convexity of the optimization problem which guarantees the uniqueness of the solution.

Another point to consider is that the goals we would like to achieve are twofold: either we have to minimize a cost function which is given by the problem itself. This can be the tolerance allowed for a device to be manufactured, the penalty for an incorrect prediction or whatever else. On the other hand we have the system generating the data we're dealing with. The output is subject to noise generated wrt. an internal noise model (e.g. roundoff errors, gaussian noise ...). In this case we can use the corresponding cost function for getting low bias and variance.

## 2.1 Maximum Likelihood Estimators

We will show the connection between a ML-estimator and the cost-function as discussed in many statistics textbooks (cfr. [Vap79]). Assuming that the data has been generated according to the following model

$$\begin{aligned} P(\vec{\mathbf{x}}, y) &= P(y|\vec{\mathbf{x}})P(\vec{\mathbf{x}}) \\ &= P(y - f(\vec{\mathbf{x}}))P(\vec{\mathbf{x}}) \end{aligned} \tag{2.1}$$

we can write the likelihood of the estimation as follows

$$P(\vec{\mathbf{x}}_1, y_1, \dots \vec{\mathbf{x}}_l, y_l) = \prod_{i=1}^{l} P(y_i - f(\vec{\mathbf{x}}_i))P(\vec{\mathbf{x}}_i) \tag{2.2}$$

writing

$$P(y - f(\vec{\mathbf{x}})) = e^{-l(y - f(\vec{\mathbf{x}}))} \tag{2.3}$$

for some function $l(\eta)$ we arrive at

$$P(\vec{\mathbf{x}}_1, y_1, \ldots \vec{\mathbf{x}}_l, y_l) = e^{-\sum\limits_{i=1}^{l} l(y_i - f(\vec{\mathbf{x}}_i))} \prod_{i=1}^{l} P(\vec{\mathbf{x}}_i) \tag{2.4}$$

hence maximizing the likelihood $P(\vec{\mathbf{x}}_1, y_1, \ldots \vec{\mathbf{x}}_l, y_l)$ results in minimizing

$$R_{ML} = \sum_{i=1}^{l} l(y_i - f(\vec{\mathbf{x}}_i)) \tag{2.5}$$

Minimizing $R_{ML}$ does not necessarily minimize the cost we have to pay due to external constraints on the system, e.g. we may be able to tolerate a deviation by $\epsilon$ without extra cost and pay only linear cost for further excess although the data we're dealing with may be normally distributed. Or in an even more extreme case the penalty for predictions may be asymmetric ($l(\eta) \neq l(-\eta)$). In this case choosing an unbiased estimator is definitely a bad choice as we are not supposed to estimate a function but to minimize a risk functional.

## 2.2 Common Density Models

We will discuss four common density models and calculate the appropriate Support Vector Regression for them. Note that these models stem from the *estimation of a location parameter* context. Howewer the distribution of errors for function estimation also depends on the function and the estimator used. Therefore the distribution of errors may differ from the distribution of the noise. The loss functions presented are robust in the corresponding class of densities [Vap79].

### 2.2.1 Gauss

Probably the most popular density model is the one of Gaussian noise (see Figure 2.1), i.e.

$$l(\eta) = \eta^2 \quad \Longrightarrow \quad P(\eta) \ \propto \ e^{-\eta^2} \tag{2.6}$$

### 2.2.2 Laplace

For systems with normal distributed variance of the data we get Laplacian noise (see Figure 2.2):

$$l(\eta) = |\eta| \quad \Longrightarrow \quad P(\eta) \ \propto \ e^{-|\eta|} \tag{2.7}$$

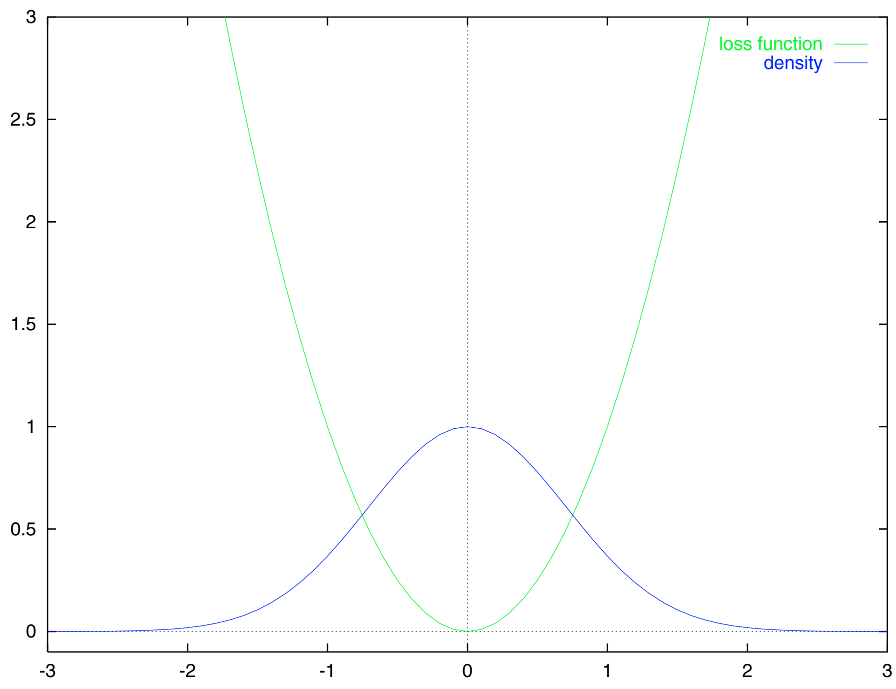


Figure 2.1: $L_2$ loss function and corresponding density

### 2.2.3 Huber

A robust estimator in the class of mixtures between a given Gaussian density and an arbitrary density symmetric w.r.t. the origin is given by the Huber loss function [Hub72], (see Figure 2.3)

$$l(\eta)_\epsilon = \begin{cases} -\eta - 0.5 & \text{for} \quad \eta < -\epsilon \\ \frac{1}{2}\eta^2 & \text{for} \quad -\epsilon \leq \eta \leq \epsilon \\ \eta - 0.5 & \text{for} \quad \eta > \epsilon \end{cases} \tag{2.8}$$

### 2.2.4 $\epsilon$-insensitive Loss Function

The loss function shown here will result in a biased estimator (unlike the ones before) when combined with a regularization term. In the realizable case it will behave like the loss function discussed in chapter 1.3, which results in an estimator like in Figure 1.1. But it has the advantage that we will not need all the input patterns for describing the regression vector $\vec{\omega}$ thereby providing a computationally more efficient way of computing the regression afterwards (see Figure 2.4).

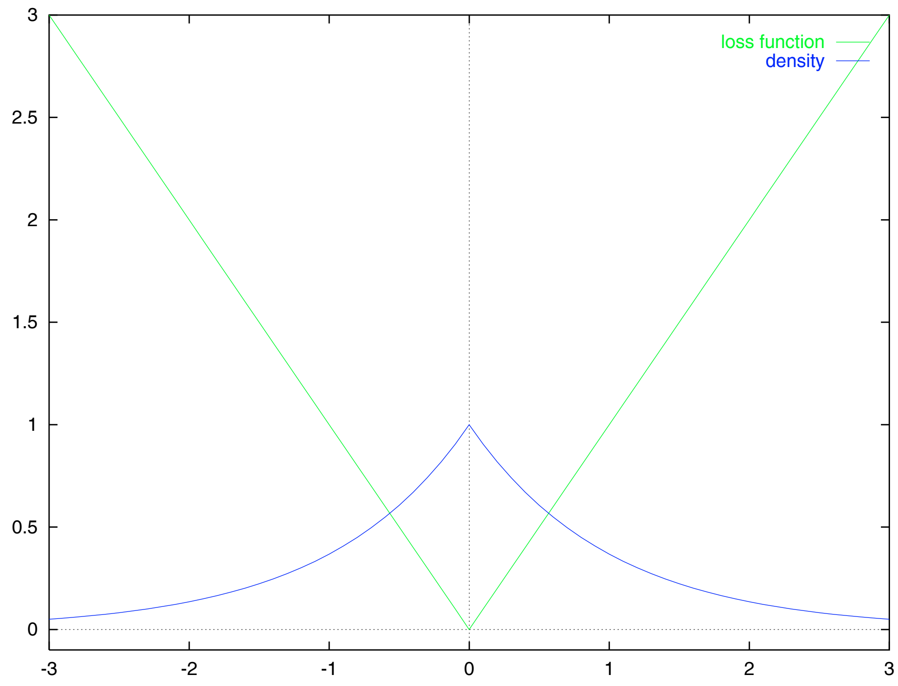


Figure 2.2: $L_1$ loss function and corresponding density

$$l(\eta)_\epsilon = \begin{cases} -\eta - \epsilon & \text{for} \quad \eta < -\epsilon \\ 0 & \text{for} \quad -\epsilon \leq \eta \leq \epsilon \\ \eta - \epsilon & \text{for} \quad \eta > \epsilon \end{cases} \tag{2.9}$$

## 2.3 Solving the Optimization Equations

We will modify equation 1.20 in a very general way to cope with indivitual loss functions $l_i(\eta)$ for each sample $(\vec{\mathbf{x}}_i, y_i)$. This can be useful for data that has been obtained by different methods or measurements that have been specified with relative precision. Our general model of a loss function $l(\eta)$ will cover functions that are zero on $[0, \epsilon]$ and convex continuous differentiable functions for $(\epsilon, \infty)$ (note: we do not require differentiability at $\epsilon$). For the sake of simplicity of notation we will use $l(\eta + \epsilon) = \zeta(\eta)$ in the following.

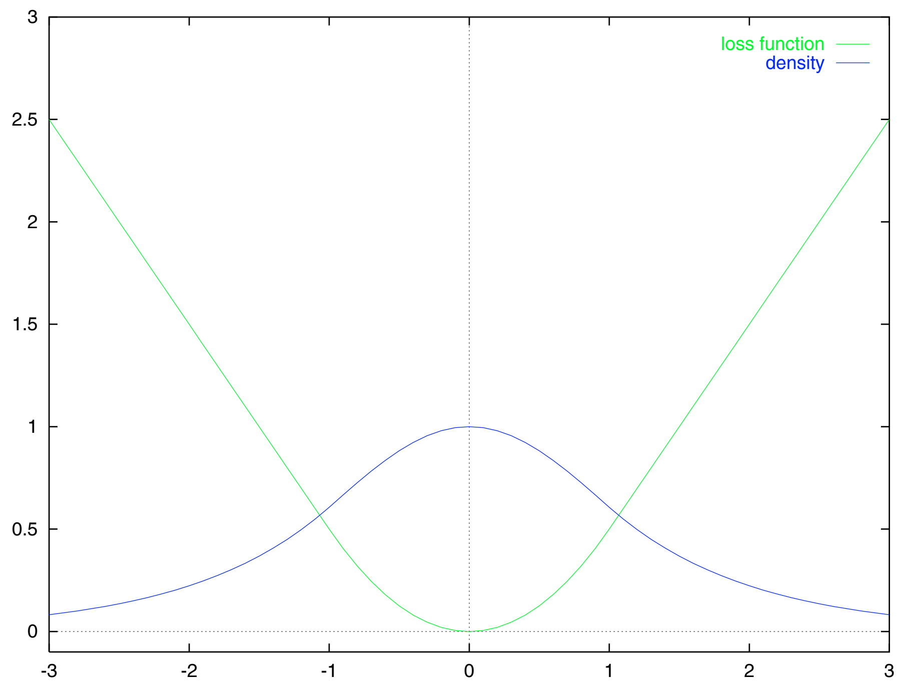


Figure 2.3: Huber loss function and corresponding density

$$
\begin{array}{ll}
\text{minimize} & R(\vec{\omega}, b, \vec{\eta}, \vec{\eta}^*) = \frac{1}{2}\|\vec{\omega}\|_2^2 + C\sum_{i=1}^{l}\left(\zeta_i(\eta_i) + \zeta_i(\eta_i^*)\right) \\
\text{subject to} & \begin{array}{rcll} f(\vec{\mathbf{x}}_i) - y_i & \leq & \eta_i + \epsilon_i & \\ y_i - f(\vec{\mathbf{x}}_i) & \leq & \eta_i^* + \epsilon_i^* & \forall\ i \in\ [1..l] \\ \eta_i, \eta_i^* & \geq & 0 & \end{array} \\
& \qquad \text{where} \quad f(\vec{\mathbf{x}}) \ = \ \vec{\omega}^t\vec{\mathbf{x}} + b
\end{array}
\tag{2.10}
$$

As in section 1.4 we can compute the Lagrange function $L$ (with dual variables $\vec{\alpha}$, $\vec{\alpha}^*$ corresponding to the constraints on $f(\vec{\mathbf{x}}_i) - y_i$ and $\vec{\gamma}$, $\vec{\gamma}^*$ corresponding to the ones on $\vec{\eta}$, $\vec{\eta}^*$. This will allow an efficient solution of the problem in the dual set of variables, especially when solving the problem in the primal set of variables will become quite intractable due to further generalizations (cfr. chapter 3).

$$
\begin{array}{rl}
L(\vec{\omega}, b, \vec{\alpha}, \vec{\alpha}^*, \vec{\eta}, \vec{\eta}^*, \vec{\gamma}, \vec{\gamma}^*) = & \frac{1}{2}\|\vec{\omega}\|_2^2 + C\sum_{i=1}^{l}\left(\zeta_i(\eta_i) + \zeta_i(\eta_i^*)\right) \\
& - \ \sum_{i=1}^{l} \alpha_i(\eta_i + \epsilon_i + y_i - f(\vec{\mathbf{x}}_i)) \\
& - \ \sum_{i=1}^{l} \alpha_i^*(\eta_i^* + \epsilon_i^* - y_i + f(\vec{\mathbf{x}}_i)) \\
& - \ \sum_{i=1}^{l} \eta_i\gamma_i + \eta_i^*\gamma_i^*
\end{array}
\tag{2.11}
$$

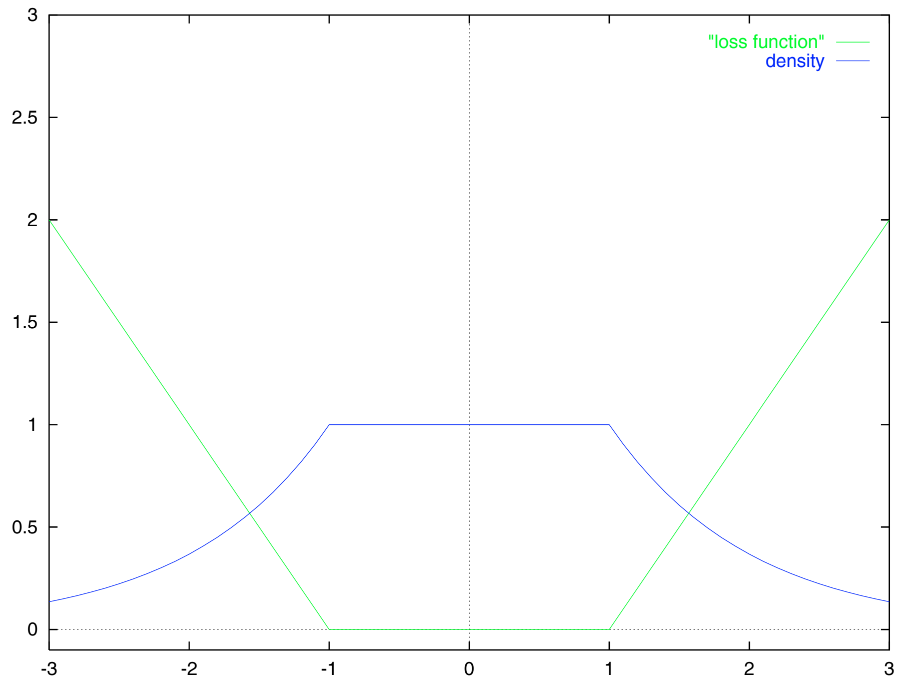


Figure 2.4: $\epsilon$-insensitive loss function and corresponding density

computing the partial derivatives of $L$

$$\begin{aligned}
\frac{\partial L}{\partial \vec{\omega}} &= \vec{\omega} - \sum_{i=1}^{l} \alpha_i \vec{\mathbf{x}}_i + \sum_{i=1}^{l} \alpha_i^* \vec{\mathbf{x}}_i && (2.12)\\
&= 0 \\
\Longrightarrow \quad \vec{\omega} &= \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \, \vec{\mathbf{x}}_i && (2.13)\\
\frac{\partial L}{\partial b} &= -\sum_{i=1}^{l} (\alpha_i - \alpha_i^*) && (2.14)\\
\Longrightarrow \quad 0 &= \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) && (2.15)\\
\frac{\partial L}{\partial \eta_i} &= C \frac{d}{d\eta_i} \zeta_i(\eta_i) - \alpha_i - \gamma_i && (2.16)\\
\frac{\partial L}{\partial \eta_i^*} &= C \frac{d}{d\eta_i^*} \zeta_i(\eta_i^*) - \alpha_i^* - \gamma_i^* && (2.17)
\end{aligned}$$

and substituting the partial derivatives back into (2.11) yields

$$\begin{aligned} W(\vec{\alpha}, \vec{\alpha}^*, \vec{\eta}, \vec{\eta}^*) &= -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) \\ & + \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i - \sum_{i=1}^{l}(\alpha_i\epsilon_i + \alpha_i^*\epsilon_i^*) \\ & + C\sum_{i=1}^{l}\left(\zeta_i(\eta_i) + \zeta_i(\eta_i^*) - \eta_i\frac{d}{d\eta_i}\zeta_i(\eta_i) - \eta_i^*\frac{d}{d\eta_i^*}\zeta_i(\eta_i^*)\right) \end{aligned} \tag{2.18}$$

Our aim is to simplify the part depending on $\zeta_i^{(*)}(\eta_i^{(*)})$ and to substitute back the corresponding choices. Without loss of generality (but helping to get rid of a lot of tedious calculus) we will only consider the terms

$$T(\eta) = \zeta(\eta) - \eta\frac{d}{d\eta}\zeta(\eta) \tag{2.19}$$

and

$$C\frac{d}{d\eta}\zeta(\eta) = \alpha + \gamma \tag{2.20}$$

with

$$\alpha, \eta, \gamma \geq 0. \tag{2.21}$$

## 2.3.1 Polynomial Loss Functions

Let us assume the general case of functions with $\epsilon$-insensitive loss zone (which may vanish) and polynomial loss of degree $p > 1$. This contains all $L_p$ loss functions as a special case ($\epsilon = 0$). The case of $p = 1$ will be treated in the following section.

$$\zeta(\eta) = \frac{1}{p}\eta^p \text{ with } p > 1 \tag{2.22}$$

$$\Longrightarrow T(\eta) = \frac{1}{p}\eta^p - \eta\eta^{p-1} = -\left(1 - \frac{1}{p}\right)\eta^p \tag{2.23}$$

$$C\eta^{p-1} = \alpha + \gamma \tag{2.24}$$

$$\Longrightarrow T(\alpha, \gamma) = -\left(1 - \frac{1}{p}\right) C^{-\frac{p}{p-1}} (\alpha + \gamma)^{\frac{p}{p-1}} \tag{2.25}$$

We want to find the maximum of $L$ in terms of the dual variables. Hence we obtain $\gamma = 0$ as $T$ is the only term where $\gamma$ appears and $T$ is maximal for that value. This yields

$$T(\alpha) = -\left(1 - \frac{1}{p}\right) C^{-\frac{p}{p-1}} \alpha^{\frac{p}{p-1}} \quad \text{with } \alpha \in \mathbb{R}_0^+ \tag{2.26}$$

Moreover from $\alpha$ we can infer how large the deviation is:

$$\eta = C^{-\frac{1}{p-1}} \, \alpha^{\frac{1}{p-1}} \tag{2.27}$$

Now for the special case of a $L_2$ loss function:

**Example 2** ($\epsilon = 0$, $p = 2$)

$$\begin{aligned} \textit{maximize } W(\vec{\alpha}, \vec{\alpha}^*) &= -\frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) && (2.28) \\ &\quad + \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) y_i - \frac{1}{2C} \sum_{i=1}^{l} ({\alpha_i}^2 + {\alpha_i^*}^2) \\ \textit{subject to} \quad & \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0 && (2.29) \\ & \alpha_i, \alpha_i^* \in [0, \infty) \end{aligned}$$

*Of course one could have obtained the solution more easily as solution of an unconstrained optimization problem which also would be computationally more efficient. The problem could be greatly simplified by combining $\alpha$ and $\alpha^*$ into $\beta = \alpha - \alpha^*$ with $\beta \in \mathbb{R}$ and $\sum_{i=1}^{l} \beta_i = 0$.*

### 2.3.2 Piecewise Polynomial and Linear Loss Function

We will use a loss function $\zeta(\eta)$ that has polynomial growth for $[0, \sigma]$, a linear one for $(\sigma, \infty)$ and is $C^1$ (one time differentiable) and convex. A consequence of the linear growth for large $\eta$ is that the range of the Lagrange multipliers becomes bounded, namely by the derivative of $\zeta$. Therefore we will have to solve box constrained optimization problems.

$$\zeta(\eta) = \begin{cases} \sigma^{1-p} \frac{1}{p} \, \eta^p & \text{for } \eta < \sigma \\ \eta + \left(\frac{1}{p} - 1\right) \sigma & \text{for } \eta \geq \sigma \end{cases} \tag{2.30}$$

$$\Longrightarrow T(\eta) = \begin{cases} -\sigma^{1-p} \left(1 - \frac{1}{p}\right) \eta^p & \text{for } \eta < \sigma \\ -\sigma \left(1 - \frac{1}{p}\right) & \text{for } \eta \geq \sigma \end{cases} \tag{2.31}$$

$$\alpha + \gamma = \begin{cases} C \, \sigma^{1-p} \, \eta^{p-1} & \text{for } \eta < \sigma \\ C & \text{for } \eta \geq \sigma \end{cases} \tag{2.32}$$

By the same reasoning as above we find that the optimal solution is obtained for $\gamma = 0$. Furthermore we can see by the convexity of $\zeta(\eta)$ that $\eta < \sigma$ iff $\alpha < C$.

Hence we may easily substitute $\alpha$ for $C$ in the case of $\eta > \sigma$. $\alpha \in [0, C]$ is always true as $\gamma \geq 0$.

$$\Longrightarrow T(\alpha) = -C^{-\frac{1}{p-1}} \left(1 - \frac{1}{p}\right) \sigma \ \alpha^{\frac{p}{p-1}} \quad \text{for all } \sigma \in \mathbb{R}_0^+ \tag{2.33}$$

Analogously (if $\alpha \in [0, C)$ we can determine the error by

$$\eta = \sigma \ C^{-\frac{1}{p-1}} \ \alpha^{\frac{1}{p-1}} \tag{2.34}$$

**Example 3 ($\epsilon$-insensitive Loss Function ($\sigma = 0$))**
*For this setting $T(\alpha)$ vanishes independently from p (left aside the case $p = 1$). This leads to the following optimization problem [Vap95].*

$$\begin{aligned} \textit{maximize}\, W(\vec{\alpha}, \vec{\alpha}^*) &= -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) \qquad (2.35) \\ &\quad + \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i - \sum_{i=1}^{l}(\alpha_i\epsilon_i + \alpha_i^*\epsilon_i^*) \\ \textit{subject to} \quad & \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \qquad (2.36) \\ & \alpha_i, \alpha_i^* \in [0, C] \end{aligned}$$

**Example 4 ($L_1$ Loss Function)**
*An even more special case is ($\epsilon = 0$, $\sigma = 0$) which we get from the equations above by removing the terms in $\epsilon$.*

**Example 5 (Relative Precision)**
*Another special case are measurements carried out with relative precision $\epsilon_{rel}$. Here we set*

$$\epsilon_i = \epsilon_i^* = \epsilon_{rel} \cdot |y_i| \tag{2.37}$$

**Example 6 (Huber Loss Function ($\epsilon = 0, p = 2$))**

$$T(\alpha) = \frac{1}{2C} \ \sigma \ \alpha^2 \tag{2.38}$$

*which results in an optimization problem of the form:*

$$\begin{aligned} \textit{maximize}\, W(\vec{\alpha}, \vec{\alpha}^*) &= -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) \qquad (2.39) \\ &\quad + \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i - \frac{1}{2C}\sum_{i=1}^{l}(\alpha_i^2\sigma_i + \alpha_i^{*2}\sigma_i^*) \\ \textit{subject to} \quad & \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \qquad (2.40) \\ & \alpha_i, \alpha_i^* \in [0, C] \end{aligned}$$

### 2.3.3 Local Loss Functions

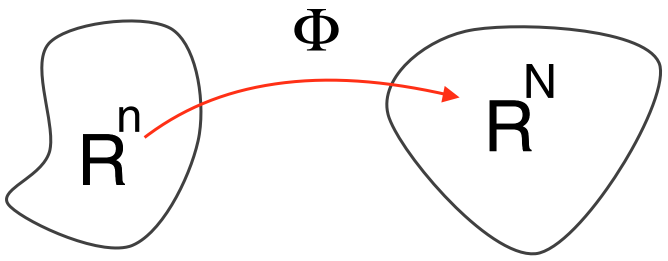
One can see easily that the loss functions described above may be combined into a local loss function. This means that when describing the regression problem one doesn't need to know the loss function over the whole domain (this may be relevant for process control systems, e.g. a vapor deposition device where for some deposition densities the accuracy may be more critical than for other ones). Instead one may define it pointwise for each sampling point. As shown there is a rich variety of functions to choose from (constant–linear, quadratic, quadratic–linear, absolute constraints on the deviation, etc.) which should provide a good approximation of the reality. One also should observe that there is no increase in computational complexity by the mixture of several cost functions, it essentially stays a quadratic programming problem.

# Chapter 3

# Kernels

## 3.1 Nonlinear Models

The problem the regression models discussed so far is that they are only linear. Instead one would like to find a general way of representing nonlinear functions in an arbitrary number of dimensions efficiently in the way described in the previous chapters. This can be achieved by a map $\Phi$ into a highdimensional space and constructing a linear regression function there. The procedure is analogous to what has been done in pattern recognition [BGV92], [ABR64].



**Example 7 (quadratic features)**

$$\begin{array}{rccl} \Phi : & \mathbb{R}^2 & \longmapsto & \mathbb{R}^3 \\ & (x_1, x_2) & \longmapsto & (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{array} \tag{3.1}$$

$\Phi$ is carried out *before* all other steps of the regression method described so far. The only modification due to this procedure is exchanging the dot products $(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)$ by $(\Phi(\vec{\mathbf{x}}_i), \Phi(\vec{\mathbf{x}}_j))$. Hence we get

$$(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) = ((x_1, x_2), (x_1^*, x_2^*)) \longmapsto ((x_1^2, \sqrt{2}x_1x_2, x_2^2), (x_1^{*2}, \sqrt{2}x_1^*x_2^*, x_2^{*2})) = (\vec{\mathbf{x}}, \vec{\mathbf{x}}^*)^2$$

This is a symmetric function K in $\vec{\mathbf{x}}$ and $\vec{\mathbf{x}}^*$. Note that in all the previous calculations $\vec{\mathbf{x}}_i$ appeared only in the form of dot products. Therefore for all the previous calculations we can easily replace $(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*)$ by $K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*)$. This gives rise to the question which general class of functions $K$ might be admissible.

## 3.2 Mercer's Condition

Using Hilbert Schmidt theory this issue can be settled in a definitive manner [CH53]: A function is called a positive Hilbert-Schmidt kernel *iff* Mercer's Condition holds:

$$\begin{aligned} \text{with } K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) &\in L_2(\mathbb{R}^n) \otimes L_2(\mathbb{R}^n) \\ \int\int_{L_2 \otimes L_2} K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) g(\vec{\mathbf{x}}) g(\vec{\mathbf{x}}^*) d\vec{\mathbf{x}} d\vec{\mathbf{x}}^* &\geq 0 \\ \forall g(\vec{\mathbf{x}}) &\in L_2(\mathbb{R}^n) \end{aligned} \tag{3.2}$$

Iff (3.2) holds true we can find an eigensystem of K $(\Phi_i(\vec{\mathbf{x}}), \lambda_i)$ with positive eigenvalues $\lambda_i$ such that K may be rewritten in the following way:

$$\begin{aligned} K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) &= \sum_i \lambda_i \Phi_i(\vec{\mathbf{x}}) \Phi_i(\vec{\mathbf{x}}^*) \qquad (3.3) \\ \text{with } \Phi_i(\vec{\mathbf{x}}^*) &= \int_\Omega K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) \Phi_i(\vec{\mathbf{x}}) d\vec{\mathbf{x}} \end{aligned}$$

In this context we may regard $\Phi_i(\vec{\mathbf{x}}_j)$ as a (potentially) infinite number features $i$ of the vector $\vec{\mathbf{x}}_j$. On the other hand any 'feature extracting'–mapping corresponds to a positive Hilbert-Schmidt kernel (proof by writing (3.2) with the 'feature extractor' as an argument).

Note that the Kernel–approach is much more powerful than just running a plain ordinary 'feature extractor' on the data as the dimensionality of the space achieved by the latter approach is only very limited whereas in the case of mapping to high dimensional spaces the implicit dimensionality may be in the order of several millions of features.

Analogously as described in section 3.1 we can substitute the dot products by the corresponding values of the kernel. One can easily see that the resulting optimization problem is still convex - this is equivalent to proving that the matrix $K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)_{(ij)}$ is positive semidefinite for any choice of $\{\vec{\mathbf{x}}_i\}$.

**Proof:**

$$\begin{aligned} & \sum_{i,j=1}^{l} K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) \alpha_i \alpha_j \qquad (3.4) \\ = & \sum_{i,j=1}^{l} \alpha_i \alpha_j \sum_k \lambda_k \Phi_k(\vec{\mathbf{x}}_i) \Phi_k(\vec{\mathbf{x}}_j) \\ = & \sum_k \lambda_k \sum_{i=1}^{l} \alpha_i \Phi_k(\vec{\mathbf{x}}_i) \sum_{j=1}^{l} \alpha_j \Phi_k(\vec{\mathbf{x}}_j) \end{aligned}$$

change of the summation order allowed as $\sum_{i,j=1}^{l}$ is just a finite sum

$$= \sum_k \lambda_k \bar{\Phi}_k \bar{\Phi}_k \ \geq \ 0 \quad \text{since } \lambda_k \geq 0 \ \ \forall k$$

## 3.3 Some Useful Kernels

- polynomial regression of order $p$

$$K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) = (1 + (\vec{\mathbf{x}}, \vec{\mathbf{x}}^*))^p \tag{3.5}$$

- RBF-kernels

$$K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) = e^{-\frac{||\vec{\mathbf{x}} - \vec{\mathbf{x}}^*||_2^2}{2\sigma^2}} \tag{3.6}$$

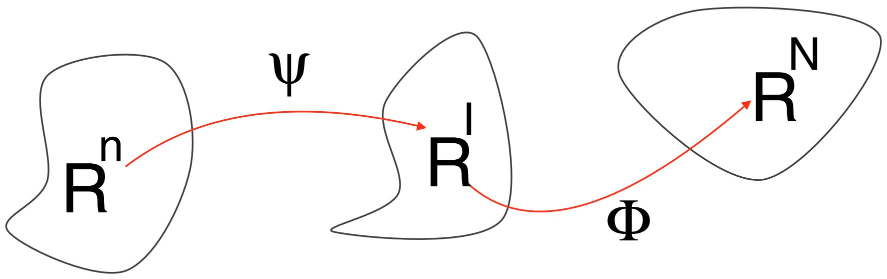
- two layer feed forward neural network

$$K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) = \tanh(\kappa(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) - \Theta) \tag{3.7}$$

These methods work pretty well for pattern recognition. In regression estimation the number of input dimensions is much lower though (compare approx. 700 pixels of an image for OCR to 3 dimensions for video data). Furthermore one would like to control the type of functions used for approximation / regression directly, e.g. one would like to construct splines. Therefore we need another method for finding a suitable kernel.

## 3.4 Intermediate Mapping

We will use an intermediate map to increase dimensionality just like in the 'preprocessing step' described in section 3.1 (this is done in contrast to Bellman's often cited curse of dimensionality). For the beginning let us suppose $n = 1$, i.e. we are operating in $\mathbb{R}^1$



Given a set of functions $\psi_k : \mathbb{R} \longmapsto \mathbb{R}$ and an admissible Hilbert-Schmidt Kernel $K_s$ on $L_2(\mathbb{R}^m) \otimes L_2(\mathbb{R}^m)$ we construct the following kernel:

$$K(x, x^*) = K_s((\psi_1(x), \ldots, \psi_m(x)), (\psi_1(x^*), \ldots, \psi_m(x^*))) \tag{3.8}$$

By construction this kernel fulfils Mercer's condition as $K_s$ does.

**Example 8 (Splines of Degree $p$)**
*Given a set $T = \{t_1, \dots t_m\}$ with $t_i < t_j$ for $i < j$ define*

$$\psi_k(x) := (x - t_k)_+^p = \begin{cases} (x - t_k)^p & \text{for} \quad x > t_k \\ 0 & \text{for} \quad x \leq t_k \end{cases} \tag{3.9}$$

*By doing this the $t_i$ define a one-dimensional grid on which spline interpolation is done. Now set $K_x(\,.\,,\,.\,) = (\,.\,,\,.\,)$. This yields*

$$K(x, x^*) = \sum_{k=1}^{m} (x - t_k)_+^p (x^* - t_k)_+^p \tag{3.10}$$

*therefore also the regression $f(x)$ becomes a spline of degree $p$*

$$\begin{aligned} f(x) &= \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) K(x, x_i) + b & (3.11) \\ &= \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \sum_{k=1}^{m} (x - t_k)_+^p (x_i - t_k)_+^p + b & (3.12) \\ &= \sum_{k=1}^{m} (x - t_k)_+^p \left( \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)(x_i - t_k)_+^p \right) + b & (3.13) \end{aligned}$$

Instead of splines we might use any other set of functions. So in a more general fashion we would have to deal with $\psi_i(x) = \psi(x - t_i)$. Choosing such a set of special functions may be convenient when solving operator equations [VGS96]. Here one could use a special $\psi$ such that $\hat{A}^{-1}\psi$ becomes a simple function.

## 3.5 Tensor Products

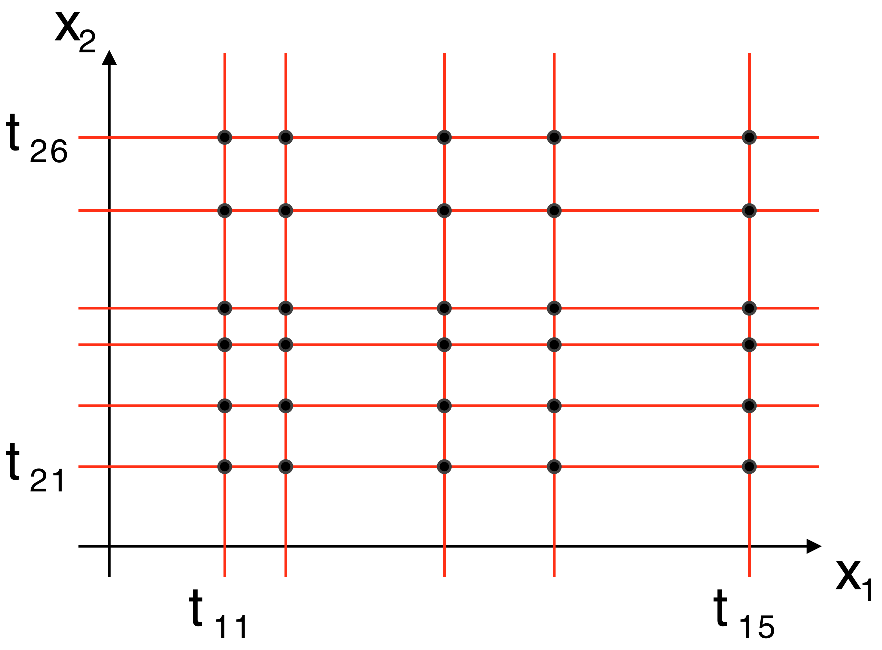
A shortcoming of this approach is that the number of functions would increase exponentionally with the dimensionality of the input space in order to provide a resolution good enough for approximating given data. Remembering that $\mathbb{R}^n = \bigotimes_{i=1}^{n} \mathbb{R}$ we will use tensor products of suitable kernels for each input dimension:

$$K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) = \prod_{i=1}^{n} K_i(x_i, x_i^*) \tag{3.14}$$

This means that we can get an n-dimensional spline expansion at only linear cost instead of an exponentional one in the number of input dimensions. Another way of describing splines would be an ANOVA-decomposition [SHKT95].

The sets $T_j = \{t_{j_1}, \dots t_{j_m}\}$ form the grid corresponding to the tensor product kernel. Note that the choice of gridpoints is independent for each dimension.

**Example 9 (Grid in Two Dimensions)**



## 3.6 Direct Mapping into Hilbert Space

Although linear in the number of dimensions the computational cost still may be very high especially for the case of choosing a very fine grid. This problem can be solved by taking the number of gridpoints per dimension to infinity thereby generating an infinitely fine grid and computing the limit analytically which yields an integral over the mapping-function.

**Proof:**
Let $K$ be normalized by the number of gridpoints $\prod_{i=1}^{n} m_i$. Setting $t_{i_k} = t_i + k \cdot \Delta t_i$ and $T_i = t_i + n_i \cdot \Delta t_i$ we get:

$$\begin{aligned} K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) &= \frac{1}{\prod_{i=1}^{n} m_i} \prod_{i=1}^{n} \sum_{k=1}^{m_i} \psi(x_i - t_{i_k}) \psi(x_i^* - t_{i_k}) \\ &= \frac{1}{\prod_{i=1}^{n} m_i \Delta t_i} \prod_{i=1}^{n} \sum_{k=1}^{m_i} \Delta t_i \psi(x_i - t_i - k \cdot \Delta t_i) \psi(x_i^* - t_i - k \cdot \Delta t_i) \end{aligned} \tag{3.15}$$

taking the limit $n_i \longrightarrow \infty$ with $n_i \Delta t_i = const.$ yields the definition of the Riemann–integral:

$$K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) \rightarrow \frac{1}{\prod_{i=1}^{n} T_i - t_i} \prod_{i=1}^{n} \int_{t_i}^{T_i} \psi(x_i - \tau_i) \psi(x_i^* - \tau_i) d\tau \tag{3.16}$$

An easier way to derive this result is to use a map into some Hilbert space $\mathcal{H}$

$$\begin{aligned} \mathbb{R}^n &\longmapsto \mathcal{H} && (3.17)\\ \vec{\mathbf{x}} &\longmapsto \psi_{\vec{\mathbf{x}}} && (3.18)\\ \Longrightarrow K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) &= (\psi_{\vec{\mathbf{x}}}, \psi_{\vec{\mathbf{x}}^*}) && (3.19) \end{aligned}$$

By construction this kernel satisfies Mercer's condition.

For simplicity we will assume n = 1 and omit the normalization

**Example 10 (Splines)**

$$\begin{aligned} \psi(\,.\,) &= (\,.\,)_+^p, \ \textit{with}\ t \le x \le y \le T && (3.20)\\ \Longrightarrow K(x,y) &= \int_t^T (x-\tau)_+^p (y-\tau)_+^p d\tau && (3.21)\\ &= \int_t^x (x-\tau)^p (y-\tau)^p d\tau && (3.22)\\ & \textit{substituting}\ \nu := x-\tau,\ \delta := y-x\ \textit{yields} && \\ &= \int_0^{x-t} \sum_{j=1}^{p} \binom{p}{j} \nu^{p+j} \delta^{p-j} d\tau && (3.23)\\ &= \sum_{j=1}^{p} \frac{1}{p+j+1} \binom{p}{j} (x-t)^{p+j+1} (y-x)^{p-j} && (3.24) \end{aligned}$$

*This expression can be evaluated several hundred times faster than summing up the single spline coefficients*[1].

**Example 11 (B-Splines)**
*We define B-Splines in a recursive manner ($\circ$ denotes the convolution operation and* **1** *the indicator function) [UAE91]:*

$$\begin{aligned} B_0 &= \mathbf{1}_{[-0.5,0.5]} && (3.25)\\ B_n &= B_{n-1} \circ B_0 \ \textit{for}\ n \ge 1 && (3.26)\\ &= \bigcirc_{i=1}^{n} B_0 && (3.27) \end{aligned}$$

*One can easily see that $B_n$-Splines (for n odd) are symmetrical positive functions in $C^{n-1}$ with compact support $[-\frac{n}{2}, \frac{n}{2}]$ (see figure 3.1). For convenience we will choose the interval $[t,T]$ sufficiently large such that for all samples $x_i$ the translated support of $B_n$ is contained in the integration interval. This is a*

[1]For sume settings also the sum may be expressed in closed form thereby decreasing computation time.

*reasonable choice as it corresponds to a translation invariant model of function approximation.*

$$\begin{aligned} K(x,y) &= \int_t^T B_n(x-\tau)B_n(y-\tau)d\tau && (3.28)\\ &= \int_{\mathbb{R}} B_n(x-\tau)B_n(\tau-y)d\tau && (3.29)\\ &= B_{2n+1}(y-x) && (3.30)\end{aligned}$$

Kernels of the $K(x,y) = f(x-y)$ type can reduce the algorithmic complexity drastically. See section 4.4.5 how the computational cost can be reduced from $O(l^3)$ to $O(l^2)$ for equidistant data and a kernel of the form described above. Moreover kernels for which $f$ has bounded support can reduce the computational effort even further by rendering the dot product matrix more diagonal. The far off–diagonal elements vanish. These kernels take into account only local correlations. This can be advantageous in the case of images [UAE91] or any other similar data (e.g. audio). See figure 3.2 for the autocorrelation function of an audio signal. One can observe that the correlation decays with the temporal distance between the measurements. Hence a kernel with support containing the alternating part of the autocorrelation could be used to approximate this kind of data. Still the generalization ability decreases until for the case that $\operatorname{supp} f$ is contained in the distance between any two adjoint points the number of nonzero Lagrange multipliers (i.e. Support Vectors) becomes equal to the number of samples (the dot product matrix $K_{ij}$ is diagonal in this case).

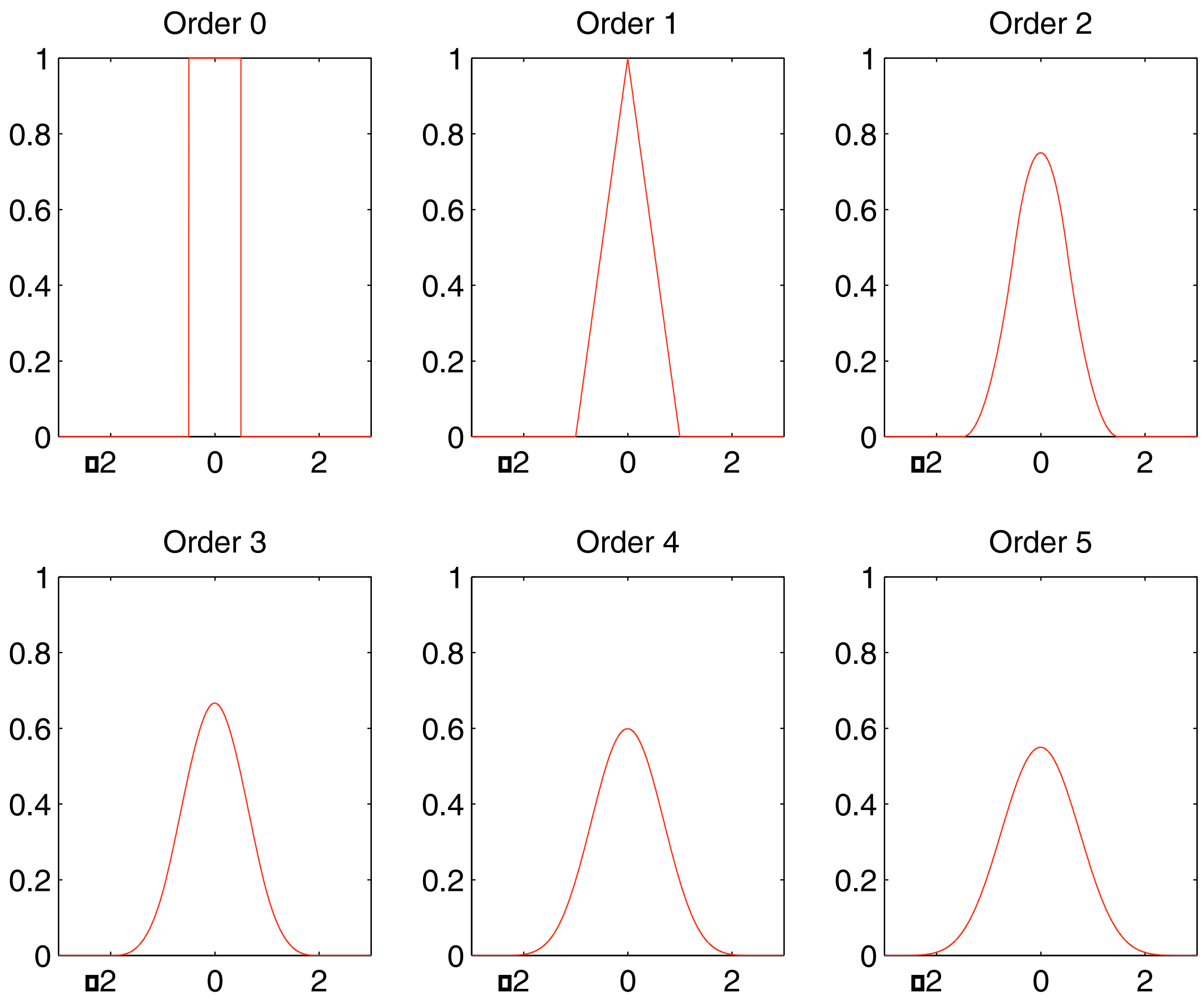


Figure 3.1: B-Splines of degree 0 up to 5

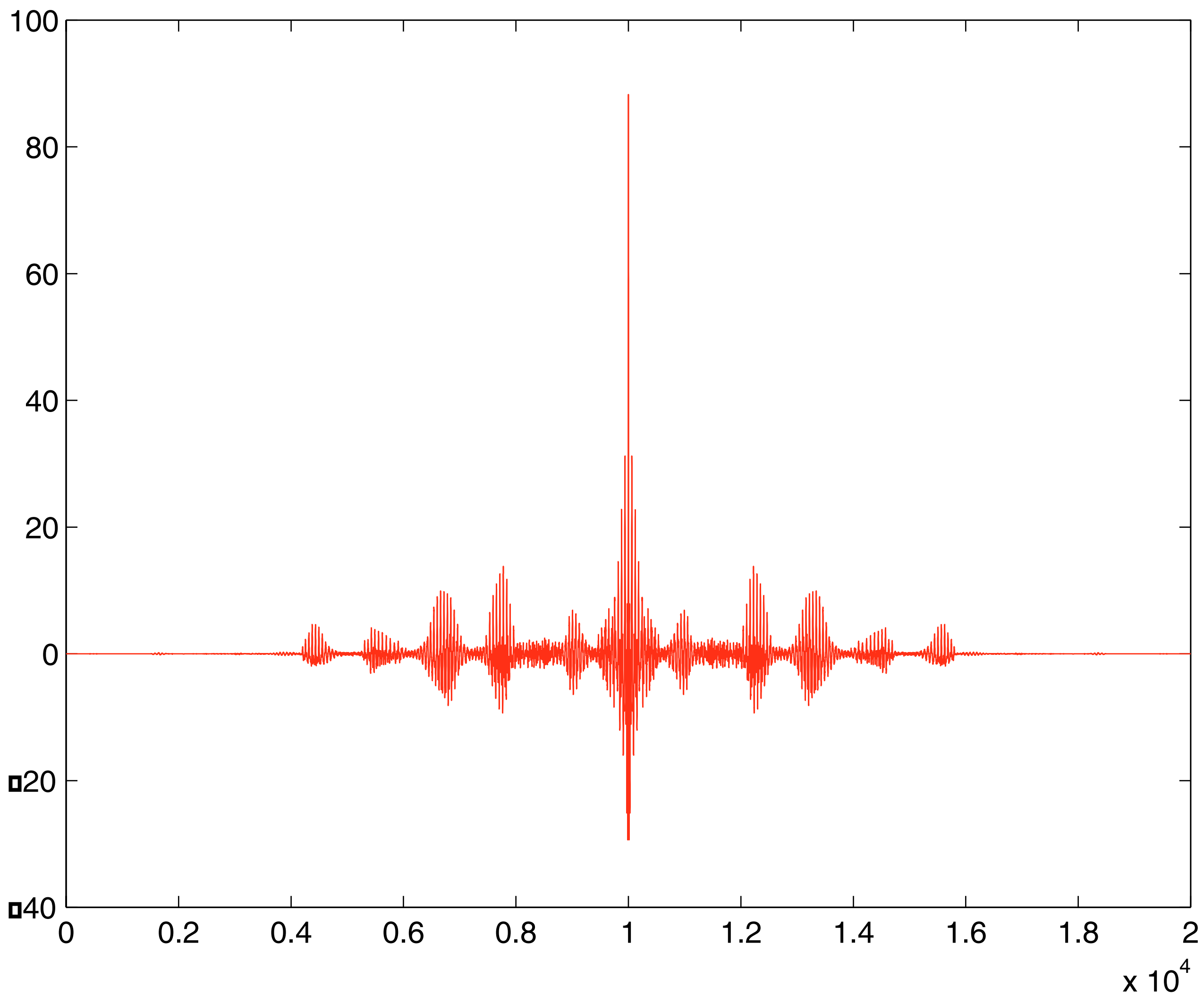


Figure 3.2: Autocorrelation Function for an audio signal: “Fischers Fritz fischt frische Fische”

# Chapter 4

# Optimizer

In this chapter we will have a closer look at optimization algorithms suitable for solving the problems posed in chapter 2. Two methods are discussed, namely a combined Conjugate Gradient and Projection method by More & Toraldo [MT91] and a special type of Interior Point Primal–Dual Path–Following algorithms in the spirit of Vanderbei [Van94]. It will be shown how for the special case of equidistant data and a restricted class of kernels the complexity of these algorithms can be reduced quite drastically by dealing with a modified Levinson algorithm. Therefore it is convenient to define a general class of optimization problems which contains the several Support Vector problems as subcases.

## 4.1 A Class of QP-Problems

$$\begin{aligned} \text{minimize } f_p(\vec{\alpha}, \vec{\alpha}^*) &= \Big(\vec{\mathbf{t}}, (\vec{\alpha}^* - \vec{\alpha})\Big) + \Big(\vec{\mathbf{t}}^*, (\vec{\alpha}^* + \vec{\alpha})\Big) \\ &\quad + \frac{1}{2}(\vec{\alpha}^* - \vec{\alpha})^t D (\vec{\alpha}^* - \vec{\alpha}) \\ &\quad + \frac{1}{2}(\vec{\alpha}^* + \vec{\alpha})^t D^* (\vec{\alpha}^* + \vec{\alpha}) \\ \text{subject to} \quad & (\vec{c}, (\vec{\alpha}^* - \vec{\alpha})) = 0 \\ & 0 \le \alpha_i, \alpha_i^* \le U \quad \forall i \in \{1..l\} \end{aligned} \tag{4.1}$$

where $D$ and $D^*$ are positive (semi)definite matrices, $\vec{\mathbf{t}}$ is an arbitrary vector and $\vec{\mathbf{t}}^*$ contains only non negative entries. The previously (in Chapter 2) described optimization problems with box constraints fall into that class of QP-problems (the other problems can be solved in a quite straightforward manner by simpler methods or by specializing the methods described here and removing the constraints in $U$ which drastically reduces the number of dual variables[1]). Additionally in

[1] $U$ coincides with $C$ of the chapters above. Still I decided to call the boundary $U$ in order to be consistent with the current implementation.

all the cases described $D^*$ only contains diagonal elements. For convenience we will make use of this fact in the following (but the reader should be aware that the corresponding statements also hold for non-diagonal $D^*$).

## 4.2 An Equivalent Formulation

For practical (and performance) reasons it is not convenient to solve the problem as is. Solving it this way would result in having to deal with a quadratic part of the type:

$$\begin{aligned} & (\vec{\alpha}^* - \vec{\alpha})^t D(\vec{\alpha}^* - \vec{\alpha}) + (\vec{\alpha}^* + \vec{\alpha})^t D^*(\vec{\alpha}^* + \vec{\alpha}) \\ = & \begin{pmatrix} \vec{\alpha} \\ \vec{\alpha}^* \end{pmatrix}^t \begin{pmatrix} D^* + D & D^* - D \\ D^* - D & D^* + D \end{pmatrix} \begin{pmatrix} \vec{\alpha} \\ \vec{\alpha}^* \end{pmatrix} \end{aligned}$$

This matrix is highly degenerate — half of its eigenvalues are zero, the other half is twice the eigenvalues of $D$. Hence we (block) diagonalize this system by switching to:

$$\begin{aligned} \vec{\beta} &= \vec{\alpha}^* - \vec{\alpha} \\ \vec{\beta}^* &= \vec{\alpha}^* + \vec{\alpha} \end{aligned} \tag{4.2}$$

This improves the condition of the problem as we don't have to rely on automatic pivoting but can perform the critical steps beforehand. The problem transforms to:

$$\begin{aligned} \text{minimize } f_p(\vec{\beta}, \vec{\beta}^*) &= (\vec{\mathbf{t}}, \vec{\beta}) + (\vec{\mathbf{t}}^*, \vec{\beta}^*) + \frac{1}{2}\vec{\beta}^t D \vec{\beta} + \frac{1}{2}\vec{\beta}^{*t} D^* \vec{\beta}^* \qquad (4.3) \\ \text{subject to } \quad & (\vec{c}, \vec{\beta}) = 0 \\ & 0 \le (\beta_i^* + \beta_i), (\beta_i^* - \beta_i) \le 2U \quad \forall i \in \{1..l\} \end{aligned}$$

Now we arrived at a sipler functional form at the cost of a more complicated combination of constraints (cf. Figure 4.1) which will be simplified now:

$$0 \le \beta_i^* + \beta_i \text{ and } 0 \le \beta_i^* - \beta_i \iff |\beta_i| \le \beta_i^* \tag{4.4}$$

$$\beta_i^* + \beta_i \le 2U \text{ and } \beta_i^* - \beta_i \le 2U \implies |\beta_i| \le U \tag{4.5}$$

We will show that choosing $\beta_i^* = |\beta_i|$ is an optimal choice:

**case 1: $D_{ii}^* = 0$ and $t_i^* = 0$**
$\beta_i^*$ does not appear neither in $f_p$ nor in the regression itself. Therefore it may take any value and $\beta_i^* = |\beta_i|$ is the best choice as it leaves $[-U, U]$ as admissible interval for $\beta_i$.

**case 2: $D_{ii}^* \ge 0$ or $t_i^* \ge 0$**
Here the optimal solution is obtained for the smallest $\beta_i^*$ possible, $|\beta_i|$.

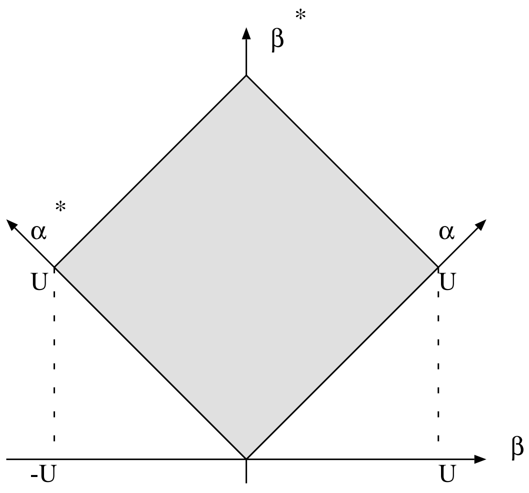


Figure 4.1: Boundary Constraints on $\alpha$ and $\beta$

All other cases have been excluded already in the definition of equation (4.1). So we can simplify the boundary conditions to:

$$\beta_i \in [-U, U] \text{ and } \beta_i^* = |\beta_i| \tag{4.6}$$

Note: this formulation is equivalent to $\alpha_i \alpha_i^* = 0$ which is obvious when looking at the formulation of the initial optimization problem (we can't exceed both the upper and the lower boundary at the same time).

This form is being used for interfacing the Support Vector Machine with an external optimizer (cf. Appendix B).

## 4.3 Classical Optimizers

The first optimizer to be used was a classical constraint based one (viz. an optimizer that is aware of the set of active / inactive boundary conditions) by More & Toraldo [MT91]. It had been successfully deployed for Support Vector Pattern Recognition problems (it is one of the fastest classical optimization currently known). I will give a short overview of the basic ideas. Its main loop consists of two alternating algorithms: projected gradient descent and conjugate gradient descent. Each of these two algorithms is iterated until no sufficient progress is made. Convergence in a finite number of steps can be proven for positive definite matrices.

### 4.3.1 Projected Gradient Descent

For the problem

$$\text{mininize } \{f_p(\vec{\mathbf{x}}) | l_i \leq x_i \leq u_i\} \tag{4.7}$$

we define a projected gradient

$$\left[\frac{\partial f_p(\vec{\mathbf{x}})}{\partial x_i}\right] := \begin{cases} \partial_i f_p(\vec{\mathbf{x}}) & \text{if } x_i \in (l_i, u_i) \\ \min(\partial_i f_p(\vec{\mathbf{x}}), 0) & \text{if } x_i = l_i \\ \max(\partial_i f_p(\vec{\mathbf{x}}), 0) & \text{if } x_i = u_i \end{cases} \tag{4.8}$$

in other words: restrict the gradient so that it does not point outside the feasible region. Now perform a line search in this direction until the minimum is found or another constraint is hit. This is guaranteed to lower the value of $f_p$ as we are dealing with a quadratic function. The advantage of this method is that more than one constraint may change per iteration thereby allowing a potentially faster convergence. When little progress is made search for the minimum on the face of the hypercube:

### 4.3.2 Conjugate Gradient Descent

Now we consider the original problem restricted to the set of variables with inactive constraints. There finding the minimum reduces to an unconstrained optimization problem (the projection on the constraints is performed afterwards) which is guaranteed to converge in at most $n$ steps (where $n$ is the dimensionality of the space):

Recursively we choose $\{\vec{\mathbf{p}}_1, \vec{\mathbf{p}}_2, \ldots\}$, $\{\vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \ldots\}$ and $\vec{\mathbf{g}}_i = \operatorname{grad}_{\vec{\mathbf{x}}} f(\vec{\mathbf{x}}_{i+1})$ with

$$\vec{\mathbf{x}}_{i+1} = \vec{\mathbf{x}}_i + \alpha_i \vec{\mathbf{p}}_i \tag{4.9}$$

such that

$$(\vec{\mathbf{p}}_j, \vec{\mathbf{g}}_i) = 0 \quad j < i \tag{4.10}$$

This means setting $\vec{\mathbf{x}}_i$ so that the gradient is orthogonal to the previous direction of descent $\vec{\mathbf{p}}_i$. One can easily see (after some calculus) that the set of $\{\vec{\mathbf{p}}_1, \vec{\mathbf{p}}_2, \ldots\}$ is linearly independent. Therefore it spans the whole space after $n$ steps and by enforcing orthogonality we get $\vec{\mathbf{g}}_{n+1} = 0$ which is a solution. In the algorithm described above not all n steps are used: when the relative improvement decreases the system will resume with projected gradient descent.

Unfortunately the algorithm was designed to work for for box constrained problems — additional constraints were taken into account by adding a large penalty term for their violation which contributed to additional computational errors. Moreover convergence had been proven for positive definite matrices only. In the regression estimation case this meant adding a (sufficiently large to ensure rapid convergence) additional diagonal regularization term which drastically deteriorated results. Hence we decided to use an interor point method instead.

## 4.4 Interior Point Methods

Interior Point methods come in three basic flavors:

**barrier methods:** Here the constraints are added as a penalty term (e.g. the logarithm of the constraints) and relaxed on convergence

**affine scaling methods:** On approaching the constraints the metric is changed. Thereby one avoids violating them.

**primal-dual path following methods:** The fact that satisfaction of the primal + dual constraints and the KKT-conditions yields the optimal solution is exploited by transforming the optimization problem into one of solving a set of equations.

In our further discussion we will only consider the latter but before doing so we will have to reformulate the optimization problem in a way that we only have positivity and linear constraints on the primal set of variables in order to efficiently obtain the dual problem.

## 4.4.1 Primal Problem

We will use the setup of equation (4.1) with a small modification of the constraints: instead of

$$0 \leq \alpha_i, \alpha_i^* \leq U \tag{4.11}$$

we will introduce a new set of slack variables $\eta_i, \eta_i^*$ and write

$$0 \leq \alpha_i, \alpha_i^*, \eta_i, \eta_i^* \quad \text{and} \quad \alpha_i^{(*)} + \eta_i^{(*)} = U \tag{4.12}$$

## 4.4.2 Dual Problem

Now we can write down the Lagrange function and compute the dual. Following the lines of [Van94] and [VDY94] we get.

$$\begin{aligned} L \;=\; & \Big(\vec{\mathbf{t}}, (\vec{\alpha}^* - \vec{\alpha})\Big) + \Big(\vec{\mathbf{t}}^*, (\vec{\alpha}^* + \vec{\alpha})\Big) \\ & + \frac{1}{2}(\vec{\alpha}^* - \vec{\alpha})^t D (\vec{\alpha}^* - \vec{\alpha}) \\ & + \frac{1}{2}(\vec{\alpha}^* + \vec{\alpha})^t D^* (\vec{\alpha}^* + \vec{\alpha}) \\ & - (\vec{\alpha}, \vec{\mathbf{a}}) - (\vec{\alpha}^*, \vec{\mathbf{a}}^*) - (\vec{\eta}, \vec{\mathbf{n}}) - (\vec{\eta}^*, \vec{\mathbf{n}}^*) \\ & - (\vec{c}, (\vec{\alpha}^* - \vec{\alpha}))\, y \\ & - ((\vec{\alpha} + \vec{\eta} - U), \vec{\mathbf{z}}) - ((\vec{\alpha}^* + \vec{\eta}^* - U), \vec{\mathbf{z}}^*) \end{aligned} \tag{4.13}$$

Differentiating wrt. the primal variables and backsubstitution yields (after some tedious algebra):

$$f_d \;=\; -\frac{1}{2}(\vec{\alpha}^* - \vec{\alpha})^t D (\vec{\alpha}^* - \vec{\alpha}) \tag{4.14}$$

$$-\frac{1}{2}(\vec{\alpha}^* + \vec{\alpha})^t D^* (\vec{\alpha}^* + \vec{\alpha})$$
$$-(U, \vec{\mathbf{n}}) - (U, \vec{\mathbf{n}}^*)$$

$$\begin{array}{rrcll} \text{subject to} & \vec{\mathbf{t}}^* + \vec{\mathbf{t}} & = & y\vec{\mathbf{c}} - \vec{\mathbf{n}}^* + \vec{\mathbf{a}}^* + (D^* + D)\vec{\alpha}^* + (D^* - D)\vec{\alpha} & (4.15) \\ & \vec{\mathbf{t}}^* - \vec{\mathbf{t}} & = & -y\vec{\mathbf{c}} - \vec{\mathbf{n}} + \vec{\mathbf{a}} + (D^* - D)\vec{\alpha}^* + (D^* + D)\vec{\alpha} & (4.16) \end{array}$$

### 4.4.3 Central Path

Finding a set of primal and dual feasible variables that fulfil the KKT-conditions is equivalent to solving (4.1). The difference between the primal and dual objective function is a measure for the quality of the current set of variables as $f_d = f$ only for the optimal solution. Unfortunately solving this system directly is a badly conditioned problem due to the KKT-conditions, hence one resorts to a method called path following which substitutes the KKT conditions with relaxed ones viz. requires them to be satisfied only approximately (4.21), (4.22) and improves the approximation iteratively. The set of equations looks as follows:

$$(\vec{c}, (\vec{\alpha}^* - \vec{\alpha})) = 0 \quad (4.17)$$
$$\alpha_i^{(*)} + \eta_i^{(*)} = U \quad (4.18)$$

$$y\vec{\mathbf{c}} - \vec{\mathbf{n}}^* + \vec{\mathbf{a}}^* + (D^* + D)\vec{\alpha}^* + (D^* - D)\vec{\alpha} = \vec{\mathbf{t}}^* + \vec{\mathbf{t}} \quad (4.19)$$
$$-y\vec{\mathbf{c}} - \vec{\mathbf{n}} + \vec{\mathbf{a}} + (D^* - D)\vec{\alpha}^* + (D^* + D)\vec{\alpha} = \vec{\mathbf{t}}^* - \vec{\mathbf{t}} \quad (4.20)$$

$$a_i^{(*)}\alpha_i^{(*)} = \mu \quad \forall i \quad (4.21)$$
$$n_i^{(*)}\eta_i^{(*)} = \mu \quad \forall i \quad (4.22)$$

$$\text{and} \quad \vec{\alpha}, \vec{\alpha}^*, \vec{\mathbf{a}}, \vec{\mathbf{a}}^*, \vec{\eta}, \vec{\eta}^*, \vec{\mathbf{n}}, \vec{\mathbf{n}}^* \geq 0 \quad (4.23)$$

($y$ is a free variable as it corresponds to an equality constraint)

### 4.4.4 Predictor / Corrector

Now we apply Newton's method for solving this system - not exactly but as a predictor/corrector algorithm [LMS90], [MS92] with computation of finite differences (e.g. $a_i \longrightarrow a_i + \Delta a_i$) and thereby linearizing the system of equations. The only problem is to decrease $\mu$ in a suitable manner such that we stay in the cone of quadratic convergence. But for the beginning let us assume that we already found a suitable value for it.

**Predictor:** compute the appropriate $\Delta_{\text{predictor}}$ for a linearized set of equations (2nd order terms are neglected) with $\mu = 0$

**Corrector:** compute a suitable value for $\mu$ (cf. 4.4.6), substitute $\Delta_{\text{predictor}}$ back into the $2^{nd}$ order terms and compute the corrector values $\Delta_{\text{corrector}}$

Hence we have to solve the following system:

$$\begin{aligned} (\vec{\mathbf{c}}, (\Delta\vec{\alpha}^* - \Delta\vec{\alpha})) &= && (4.24)\\ -(\vec{\mathbf{c}}, (\vec{\alpha}^* - \vec{\alpha})) &=: \gamma_c \\ \Delta\alpha_i^{(*)} + \Delta\eta_i^{(*)} &= && (4.25)\\ U - \alpha_i^{(*)} - \eta_i^{(*)} &=: \gamma_{u^{(*)}} \end{aligned}$$

$$\begin{aligned} \Delta y\vec{\mathbf{c}} - \Delta\vec{\mathbf{n}}^* + \Delta\vec{\mathbf{a}}^* + (D^* + D)\Delta\vec{\alpha}^* + (D^* - D)\Delta\vec{\alpha} &= && (4.26)\\ \vec{\mathbf{t}}^* + \vec{\mathbf{t}} - y\vec{\mathbf{c}} + \vec{\mathbf{n}}^* - \vec{\mathbf{a}}^* - (D^* + D)\vec{\alpha}^* - (D^* - D)\vec{\alpha} &=: \gamma_{\alpha^*} \\ -\Delta y\vec{\mathbf{c}} - \Delta\vec{\mathbf{n}} + \Delta\vec{\mathbf{a}} + (D^* - D)\Delta\vec{\alpha}^* + (D^* + D)\Delta\vec{\alpha} &= && (4.27)\\ \vec{\mathbf{t}}^* - \vec{\mathbf{t}} + y\vec{\mathbf{c}} + \vec{\mathbf{n}} - \vec{\mathbf{a}} - (D^* - D)\vec{\alpha}^* - (D^* + D)\vec{\alpha} &=: \gamma_{\alpha} \end{aligned}$$

$$\begin{aligned} \Delta a_i^{(*)} + \Delta\alpha_i^{(*)} a_i^{(*)} \alpha_i^{(*)-1} &= && (4.28)\\ \mu\alpha_i^{(*)-1} - a_i^{(*)} - \Delta\alpha_i^{(*)}\Delta a_i^{(*)}\alpha_i^{(*)-1} &=: \gamma_{a^{(*)}} \\ \Delta n_i^{(*)} + \Delta\eta_i^{(*)} n_i^{(*)} \eta_i^{(*)-1} &= && (4.29)\\ \mu\eta_i^{(*)-1} - n_i^{(*)} - \Delta\eta_i^{(*)}\Delta n_i^{(*)}\eta_i^{(*)-1} &=: \gamma_{n^{(*)}} \end{aligned}$$

In order to avoid additional tedious notations we define[2]:

$$\begin{aligned} n^{(*)} &:= \operatorname{diag}(n_1^{(*)}, \ldots n_l^{(*)}); && (4.30)\\ & \quad (a,\ \alpha \text{ and } \eta \text{ are defined analogously}) \\ \delta^{(*)} &:= n^{(*)}\eta^{(*)-1} + a^{(*)}\alpha^{(*)-1} && (4.31)\\ \rho_c &:= -\gamma_c && (4.32)\\ \rho_{\alpha^{(*)}} &:= -\gamma_{\alpha^{(*)}} + \gamma_{a^{(*)}} - \gamma_{n^{(*)}} + \gamma_{u^{(*)}} n^{(*)}\eta^{(*)-1} && (4.33) \end{aligned}$$

Equation (4.25) can be solved for $\eta_i^{(*)}$ directly. Substituting (4.28) and (4.29) into (4.27) and (4.26) yields:

$$\mathbf{M}\begin{pmatrix} \Delta\vec{\alpha}^* \\ \Delta\vec{\alpha} \\ \Delta y \end{pmatrix} = \begin{pmatrix} \rho_{\alpha^*} \\ \rho_{\alpha} \\ \rho_c \end{pmatrix} \tag{4.34}$$

with

$$\mathbf{M} = \begin{pmatrix} D^* + D + \delta^* & D^* - D & -\vec{\mathbf{c}} \\ D^* - D & D^* + D + \delta & \vec{\mathbf{c}} \\ -\vec{\mathbf{c}}^t & \vec{\mathbf{c}} & 0 \end{pmatrix} \tag{4.35}$$

[2]The abbreviation $(*)$ means that the equation holds for the variables with and without $*$.

we can simplify the system further by applying:

$$\mathbf{T} = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{hence} \quad \mathbf{T}^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.36}$$

$$\mathbf{T}^t\mathbf{M}\mathbf{T} = \begin{pmatrix} 4D^* + \delta^* + \delta & \delta - \delta^* & 0 \\ \delta - \delta^* & 4D + \delta^* + \delta & 2\vec{\mathbf{c}} \\ 0 & 2\vec{\mathbf{c}}^t & 0 \end{pmatrix} \tag{4.37}$$

The difficult part is to diagonalize $4D + \delta^* + \delta$, the rest of the blocks is easy as these contain only diagonal terms (that was the condition we imposed on $D^*$ before). Note that these blocks have full rank as the $\delta^* + \delta$ terms are guaranteed to be positive by definition.

Usually $D$ will be a symmetric positive matrix without any further properties. For the case of input data $\vec{\mathbf{x}}_i$ with equal spacing

$$\vec{\mathbf{x}}_i - \vec{\mathbf{x}}_j = (i-j)\Delta\vec{\mathbf{x}} \tag{4.38}$$

and a translation invariant Kernel (e.g. B-Splines)

$$K(\vec{\mathbf{x}}, \vec{\mathbf{x}}^*) = K(\vec{\mathbf{x}} - \vec{\mathbf{x}}^*) \tag{4.39}$$

$D$ becomes a Toeplitz matrix. Unfortunately the $\delta^* + \delta$ terms partly spoil the symmetry. So one has to modify the standard diagonalization algorithms:

### 4.4.5 Modified Levinson Decomposition

We will follow the reasoning in [PTVF92] with some modifications: Denote $R_0 + d_{ii}$ the varying diagonal elements of the (otherwise) Toeplitz matrix and $R_j$ the side diagonal elements with $R_j = R_{-j}$, viz. we have a symmetric matrix. Note that the following reasoning is valid for non symmetric matrices, too but we will not comment on that any further.

We want to solve:

$$\begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{pmatrix} = \begin{pmatrix} R_0 + d_{11} & R_1 & \cdots & R_{n-1} \\ R_1 & R_0 + d_{22} & \cdots & R_{n-2} \\ \cdot & & \cdot & \cdot \\ \cdot & & \cdot & \cdot \\ \cdot & & \cdot & \cdot \\ R_{n-1} & R_{n-2} & \cdots & R_0 + d_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix} \tag{4.40}$$

$d_{ii}$ are the additional diagonal terms stemming from the interior point method whereas $R_i = K(\vec{\mathbf{x}}_j, \vec{\mathbf{x}}_{j+i})$ are the regular entries of the dot product matrix. The key idea is to find a recursive way of diagonalizing the matrices and compute the updates $M \longmapsto M + 1$ with only $O(M)$ computational cost.

Recursion over $M$ (RHS solutions): $x_i^M$ indicates an element of the solution vector $x_i$ for the $M$-th step.

$$\begin{aligned} \text{start } (M)\text{: } y_i &= d_{ii} x_i^M + \sum_{j=1}^{M} R_{i-j} x_j^M && (4.41) \\ \text{recursion } (M+1)\text{: } y_i &= d_{ii} x_i^{M+1} + \sum_{j=1}^{M} R_{i-j} x_j^{M+1} + R_{i-M-1} x_{M+1}^{M+1} && (4.42) \\ \Longrightarrow R_{i-(M+1)} x_{M+1}^{M+1} &= d_{ii}(x_i^M - x_i^{M+1}) + \sum_{j=1}^{M} R_{i-j}(x_j^M - x_j^{M+1}) && (4.43) \end{aligned}$$

now we divide by $x_{M+1}^{M+1}$ and substitute

$$G_j^M := \frac{x_j^M - x_j^{M+1}}{x_{M+1}^{M+1}} \tag{4.44}$$

this yields

$$R_{i-(M+1)} = d_{ii} G_i^M + \sum_{j=1}^{M} R_{i-j} G_j^M \tag{4.45}$$

specializing (4.42) for $i = M+1$ and substituting $x$ for $G$ (4.44) we get

$$\begin{aligned} y_{M+1} &= \sum_{j=1}^{M} R_{M+1-j} x_j^{M+1} + (R_0 + d_{M+1,M+1}) x_{M+1}^{M+1} && (4.46) \\ x_{M+1}^{M+1} &= \frac{\sum_{j=1}^{M} R_{M+1-j} x_j^M - y_{M+1}}{\sum_{j=1}^{M} R_{M+1-j} G_{M+1-j}^M - (R_0 + d_{M+1,M+1})} && (4.47) \end{aligned}$$

This is a recursive relation in $x$ and $G$. If we could express $x$ by $G$ only we would have achieved our goal. Now we write the same set of equations for the left hand side solutions (using $z$ instead of $x$):

$$\begin{aligned} y_i &= d_{ii} z_i^M + \sum_{j=1}^{M} R_{j-i} z_j^M && (4.48) \\ \Longrightarrow R_{M+1-i} &= d_{ii} H_i^M + \sum_{j=1}^{M} R_{j-i} H_j^M && (4.49) \end{aligned}$$

with

$$H_j^M = \frac{z_{M+1-j}^M - z_{M+1-j}^{M+1}}{z_{M+1}^{M+1}} \tag{4.50}$$

as $H_{M+1}$ obeys to the same type of equations as $x$ (and $G_{M+1}$ corresponds to $z$) we can write a system of recursion equations:

$$\begin{aligned} H_{M+1}^{M+1} &= \frac{\sum_{j=1}^{M} R_{M+1-j} H_j^M - R_0}{\sum_{j=1}^{M} R_{M+1-j} G_{M+1-j}^M - (R_0 + d_{M+1,M+1})} && (4.51) \\ G_{M+1}^{M+1} &= \frac{\sum_{j=1}^{M} R_{j-M-1} G_j^M - R_0}{\sum_{j=1}^{M} R_{j-M-1} H_{M+1-j}^M - (R_0 + d_{M+1,M+1})} && (4.52) \\ G_j^{M+1} &= G_j^M - G_{M+1}^{M+1} H_{M+1-j}^M && (4.53) \\ H_j^{M+1} &= H_j^M - H_{M+1}^{M+1} H_{M+1-j}^M && (4.54) \end{aligned}$$

With the initial values

$$x_1^1 = \frac{y_1}{R_0 + d_{00}} \qquad G_1^1 = \frac{R_{-1}}{R_0 + d_{00}} \tag{4.55}$$

$$z_1^1 = \frac{y_1}{R_0 + d_{00}} \qquad H_1^1 = \frac{R_1}{R_0 + d_{00}} \tag{4.56}$$

we can start solving the system iteratively. As $R_i = R_{-i}$ we only need $G$ and can forget about $H$ for the solution. This gives a method that scales only $O(M^2)$ for inverting the matrix compared to $O(M^3)$ for standard diagonalization methods for arbitrary symmetric matrices. A further performance improvement might be obtained by modifying the $O(M \log M)$ methods of Bunch, Parlett and de Hoog [Bun85], [dH87], [BBdHS95].

### 4.4.6 Computing $\mu$ and other updates

After obtaining the update rules by the predictor / corrector steps one has to determine the length of the update steps. So far we did not care about the positivity constraints of the variables - now we will have to restrict the steplength $s$ such that no variable becomes negative:

$$\frac{1}{s} = \frac{-1}{1+\varepsilon} \min \left( \frac{\Delta \alpha_i^{(*)}}{\alpha_i^{(*)}}, \frac{\Delta a_i^{(*)}}{a_i^{(*)}}, \frac{\Delta \eta_i^{(*)}}{\eta_i^{(*)}}, \frac{\Delta n_i^{(*)}}{n_i^{(*)}}, -(1+\varepsilon) \right) \tag{4.57}$$

Setting $\varepsilon = 0.05$ is a good choice.

Another heuristic is applied to the choice of $\mu$. First we have to find a value corresponding to the current set of variables $\alpha, a, \eta, n$ for the case that the current set of variables does not lie on the central path[3]. Using equations (4.21) and (4.22) and summing up over them we get

$$\mu = \frac{(\vec{\alpha}, \vec{\mathbf{a}}) + (\vec{\alpha}^*, \vec{\mathbf{a}}^*) + (\vec{\eta}, \vec{\mathbf{n}}) + (\vec{\eta}^*, \vec{\mathbf{n}}^*)}{4l} \tag{4.58}$$

[3]The central path is the set of primal and dual feasible points for which the modified KKT–conditions hold.

As we want to decrease $\mu$ after each iteration we choose $\mu_{new}$ according to [Van94]

$$\mu_{new} = \mu_{old} \left( \frac{1-s}{1+10s} \right)^2 \tag{4.59}$$

This ensures slow decrease of $\mu$ when the problem becomes difficult (i.e. we're hitting the positivity constraints).

For monitoring the progress of the optimizer it is convenient to define some feasibility criteria $I_p$ and $I_d$ for the primal and dual set. The violation of the constraints is stored in the temporary variables $\gamma_c$ and $\gamma_{u^{(*)}}$ for the primal constraints and in $\gamma_{\alpha^{(*)}}$ for the dual ones. These have to become zero (or at least close to it) for a feasible point to be found. Hence we set

$$I_p := \frac{\sqrt{\gamma_c^2 + ||\gamma_u||_2^2 + ||\gamma_{u^*}||_2^2}}{2lU} \tag{4.60}$$

$$I_d := \frac{\sqrt{||\gamma_\alpha||_2^2 + ||\gamma_{\alpha^*}||_2^2}}{2||\vec{\mathbf{c}}||_2 + 1} \tag{4.61}$$

and declare a solution primal / dual feasible if $I$ is smaller than $10^{-6}$.

Moreover we would like to measure the feasibility gap, the difference between primal and dual objective function and call the logarithm of the relative difference number of significant figures $n_s$:

$$n_s := -\log_{10} \frac{|f_p - f_d|}{|f_p| + 1} \tag{4.62}$$

We stop if $n_s > 6$.

The starting point is chosen such that the feasibility conditions are fulfilled as well as possible. This is done in one step by solving the primal and dual feasibility conditions (4.17 - 4.20) and restricting the solution to the domain $[0.1U, \infty)$.

# Chapter 5

# Experiments

We will show the properties of Support Vector Regression Estimation using a simple function to be approximated

$$f(x) = \frac{\sin x}{x} + \xi \tag{5.1}$$

with $\xi$ as additional noise term of variable strength and distribution. For convenience set $[0,\, 1]^2$ as domain of interest and apply a scaling to $f$ so that we're effectively dealing with $0.9 \cdot \mathrm{sinc}(\frac{10}{\pi}x)$.

## 5.1 Approximation

There are several criteria with respect to which an approximation for some given dataset could be chosen. Common choices are the $L_p$-norms with $p \in \{1, 2\}$. Still this is not a safe choice as these norms are not equivalent, hence convergence in $L_p$ doesn't imply convergence in $L_{\bar{p}}$ (for $p \in \mathbb{R}^+$). The only norm that implies convergence in all other norms is $L_\infty$. This may be a serious issue in practical applications such as predicting the stock market: besides maximizing revenues one would like to keep the maximum loss (e.g. by an erroneous prediction of the system) bounded (viz. to bound it in the $L_\infty$-norm). Therefore we choose

$$||f - f_{approx}||_\infty \leq \epsilon \tag{5.2}$$

and among this class of functions satisfying (5.2) the flattest one for which this equation holds. See chapter 1 for more details.

This approach corresponds to the graphs in figure 5.1. One may observe that the approximations become flatter as the margin widens until they result in the constant function which is the flattest function available. Unfortunately in reality we are not given $f$ itself but only measurements at positions $\{\vec{\mathbf{x}}_1, \ldots \vec{\mathbf{x}}_l\}$ for which we try to fulfil the condition stated above. Effectively we have to find the flattest function that fits through the errorbars shown in the graph.

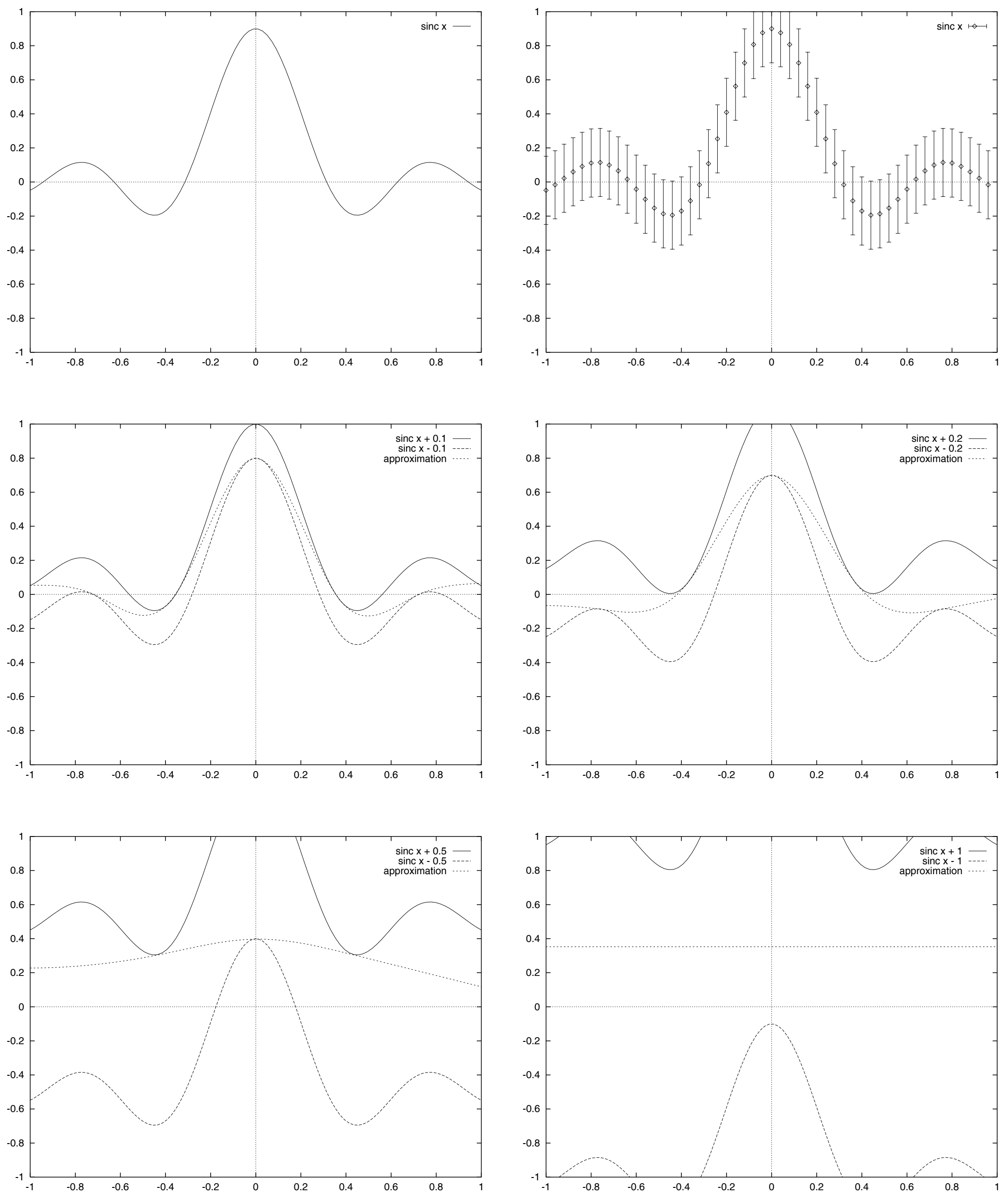


Figure 5.1: sinc $x$, datapoints with $\epsilon$-precision and approximations with 0.1, 0.2, 0.5 and 1.0 precision using a spline kernel of degree 3

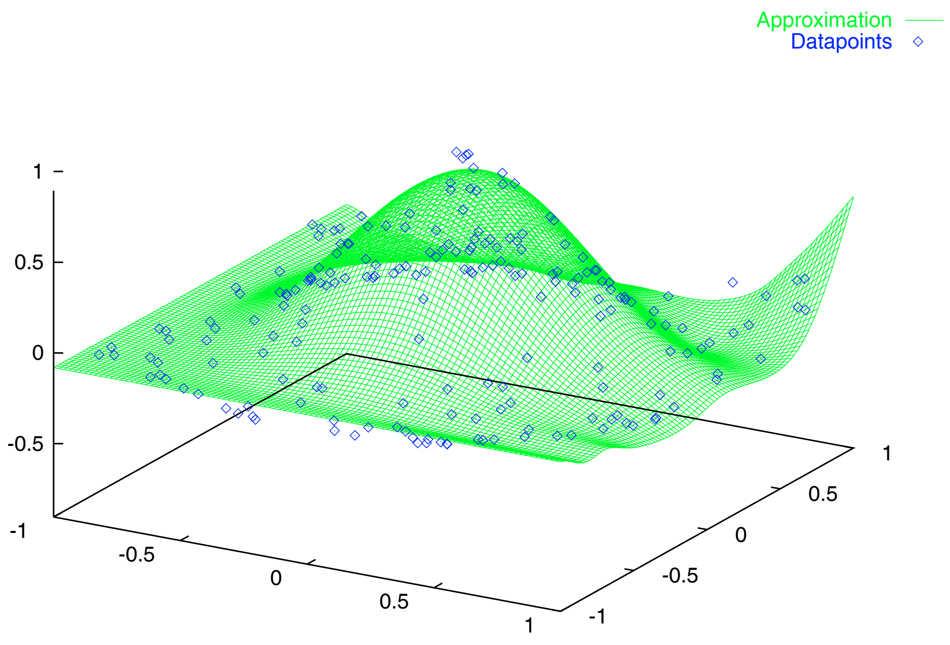


Figure 5.2: sinc $||x||$ measured at 100 iid locations with a spline approximation of degree 1 and precision 0.1

Of course such an approximation can be carried out for more than one dimension (see figure 5.2) but we will stick with just one dimension in the following as the results can be visualized much more easily.

Figure 5.4 shows the decomposition of the approximation in support vectors. One can observe that the number of support vectors increases with the quality of approximation required. This effect can be understood by a simple physical reasoning - keeping in mind that the lagrange multipliers can be regarded as *forces* and the approximation corresponds to a flexible rod one would like to fit into the $\epsilon$-*tube* it is clear that by narrowing the width of the tube the number of points where the rod hits the tube increases. In other words: by narrowing the constraints on the approximation more and more of them are likely to become active.

The action of the Lagrange multipliers can be observed directly in figure 5.3. Only at the points where the approximation hits the boundary Support Vectors appear. This is a restatement of the Karush–Kuhn–Tucker conditions (cf. section 1.5). This effect also can be seen in the context of data compression: for representing the approximation we only have to store the Support Vectors and the corresponding Lagrange multipliers and may forget about the other datapoints. As already shown before approximation is possible with computational cost scaling only in the number of samples for any dimension. This is much better than direct approximation by splines as the latter scales exponentionally in

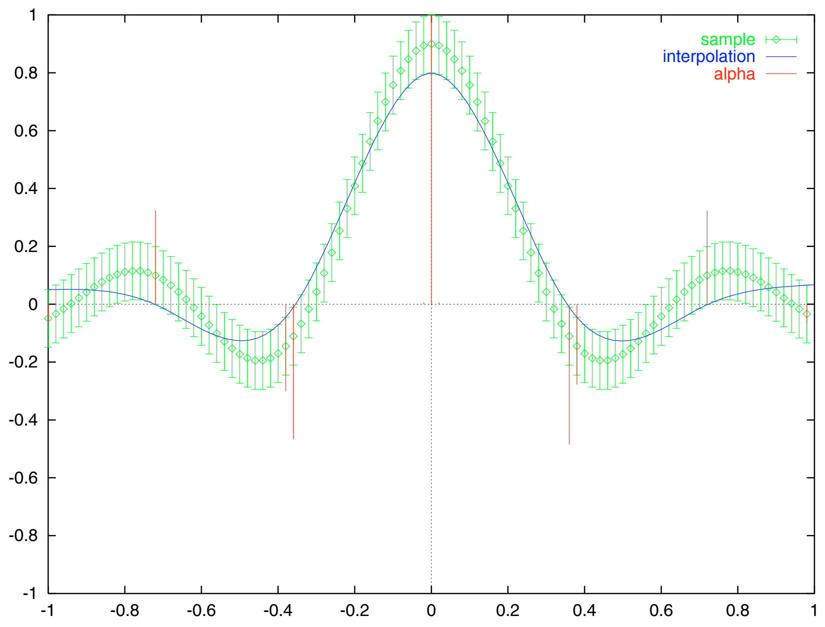


Figure 5.3: forces (red) exerted by the $\epsilon$-tube (green) on the approximation (blue)

the number of input-dimensions (viz. the number of coefficients for the function representation).

### 5.1.1 Number of Support Vectors

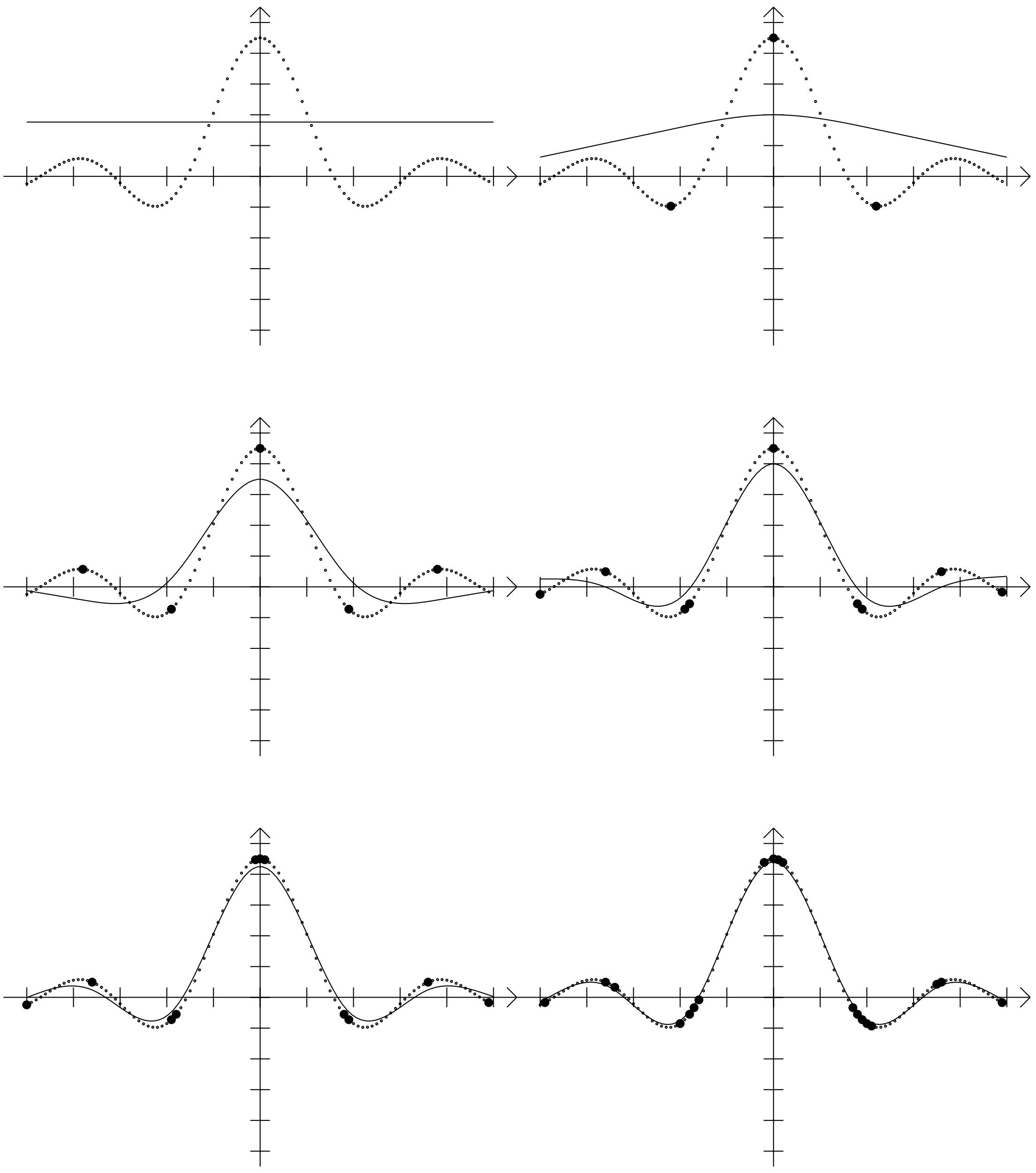
In figure 5.4 one can see once more the increase in the number of Support Vectors for increasing approximation quality, this time for an approximation with B-Splines. One can see that the quality of the approximation does not depend strongly on the kernels used. This is a similar finding to what is the case in pattern recognition. See [SBV95] for more details on this topic.

Figure 5.5 finally is an explicit compression diagram for our dataset. For approximating the 100 points given up to a certain precision $\epsilon$ we need as much data as we get Support Vectors.

## 5.2 Regression / Estimation

Regression / Estimation means trying to infer the correct functional dependency from noisy data. Hence we will turn on $\xi$ in (5.1) in the following experiments. Figure 5.6 shows the approximation properties for different noise strength.

It is interesting to see how the number of support vectors for a fixed margin increases with increasing noise. In a data compression context this means that



1.2

Figure 5.4: approximation with 1.0, 0.5, 0.2, 0.1, 0.05 and 0.02 precision precision using a B-spline kernel of degree 3, the solid dots are support vectors

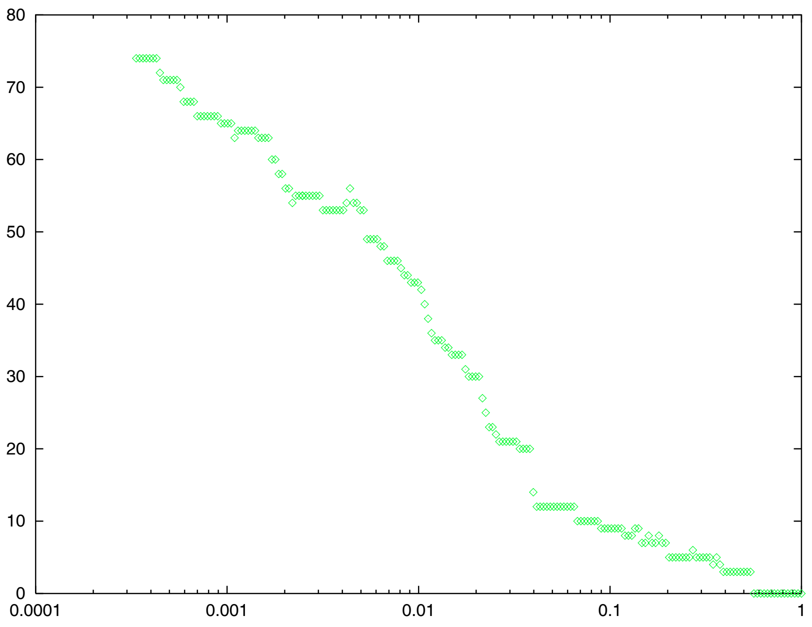


Figure 5.5: Number of Support Vectors vs. precision of the approximation (spline approximation of degree 1 with 100 nodes) for 100 datapoints (note that the maximum number of points is 100)

the more noisy data becomes the less it can be compressed. Moreover $P(|\xi| \geq \epsilon)$ is the probability of a point to exceed the $\epsilon$-margin for an unbiased estimator.

### 5.2.1 Number of Support Vectors

This increase also can be understood by a different reasoning. As the number of support vectors corresponds to the number of samples that lie on or outside the given margin one can relate this to the probability that for an unbiased estimator with sample size $l \longrightarrow \infty$ the deviation between the data and the estimate is greater or equal to $\epsilon$:

$$\begin{aligned} P(|y_i - f_{approx}(\vec{\mathbf{x}}_i)| \geq \epsilon) &= \int\limits_{(-\infty,-\epsilon]\cup[\epsilon,\infty)} p(\xi) d\,\xi && (5.3) \\ &= 2\int\limits_{\epsilon}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\xi^2}{2\sigma^2}} d\,\xi \\ &= 1 - \operatorname{erf}(\frac{\epsilon}{\sigma}) && (5.4) \end{aligned}$$

Here $\operatorname{erf}(x) := 2\frac{1}{\sqrt{2\pi\sigma^2}} \int_0^{\epsilon} e^{-\frac{\xi^2}{2\sigma^2}} d\,\xi$. Figure 5.7 shows the good agreement between (5.3) and the experimental results for fixed noise and varying $\epsilon$-margin. In figure

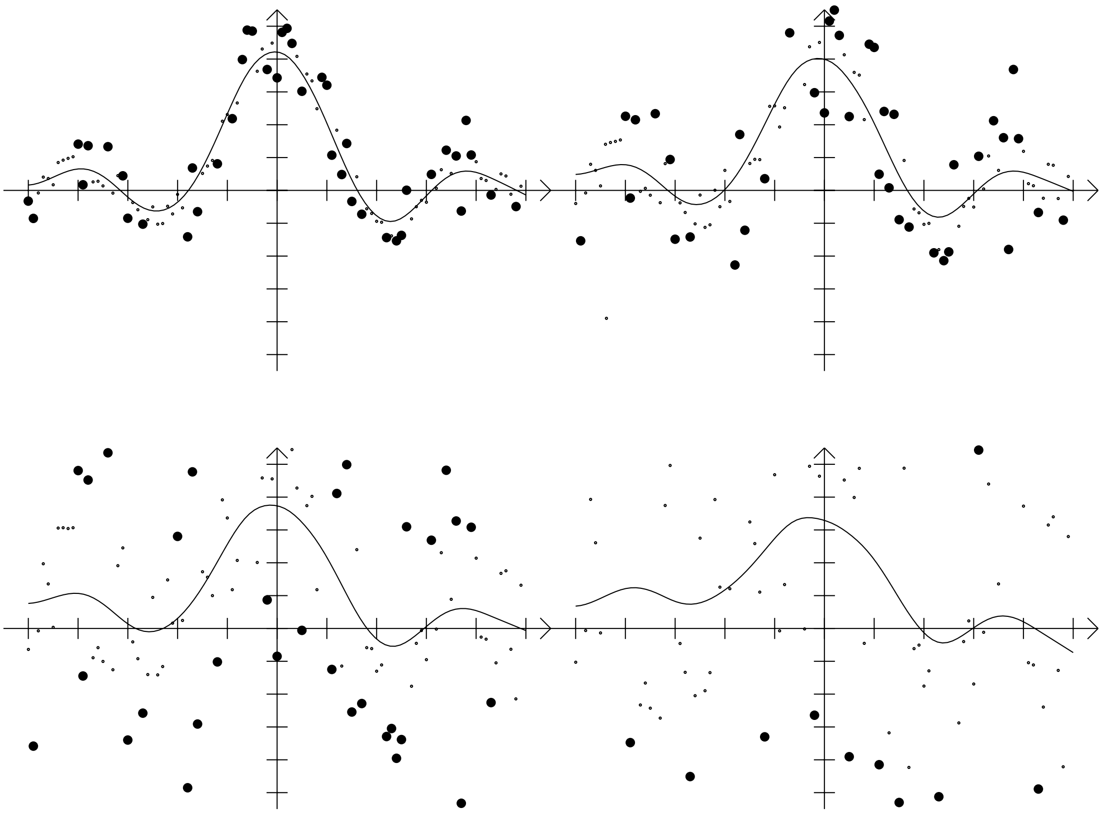


Figure 5.6: regression / estimation of $0.9\,\mathrm{sinc}\,(10x)$ on $[0,\ 1]^2$ for 100 samples with gaussian noise of standard deviation 0.1, 0.2, 0.5 and 1.0 and the corresponding $\epsilon$-insensitive loss-zone of width 0.1, 0.2, 0.5 and 1.0 (black dots are support vectors)

5.8 the same effect can be observed for fixed $\epsilon$-margin and varying gaussian noise ($\sigma$). Here howewer the number of Support Vectors does not converge to 0 for $\sigma \longrightarrow 0$ because the function without noise still carries information. This is the reason for the discrepancy between P(SV) and the measured number of Support Vectors. The same is true for the region $\epsilon \in [0.5, 1]$ for figure 5.7.

## 5.2.2 Optimal Choice of Parameters

The approach described so far leaves two questions unanswered: The choice of $\epsilon$ and the choice of the regularization. It can be seen from the figures 5.11, 5.12, 5.13, 5.14 that there is (not surprisingly) an optimal choice for both variables. Moreover one will notice that the smallest error is not obtained for $\epsilon = 0$ but for some value greater than zero. This means that the $\epsilon$-insensitive loss function that corresponds to a biased estimator performs better than the unbiased $L_1$ counterpart for gaussian noise.

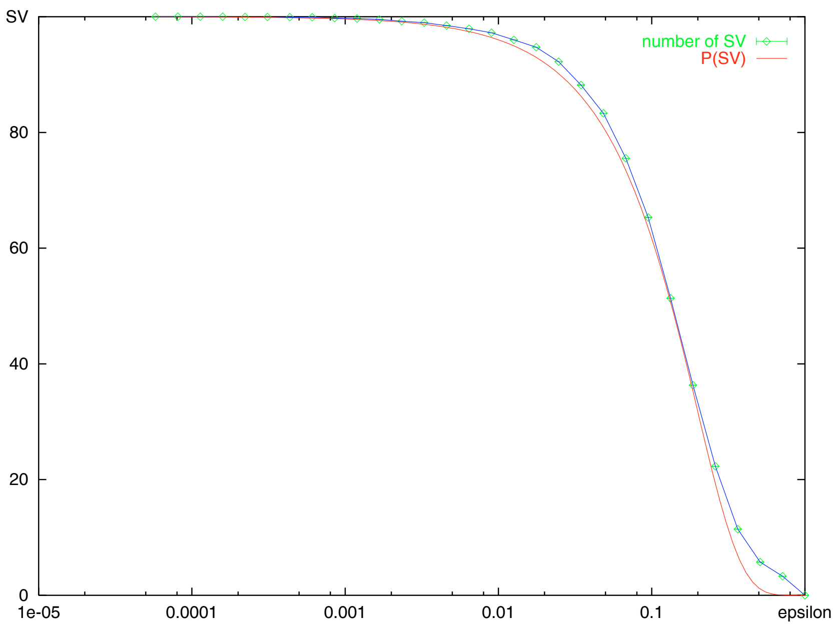


Figure 5.7: Number of Support Vectors depending on the $\epsilon$-margin for fixed noise level and probability of a point to lie outside the margin for an unbiased estimator and noise level $\sigma = 0.2$

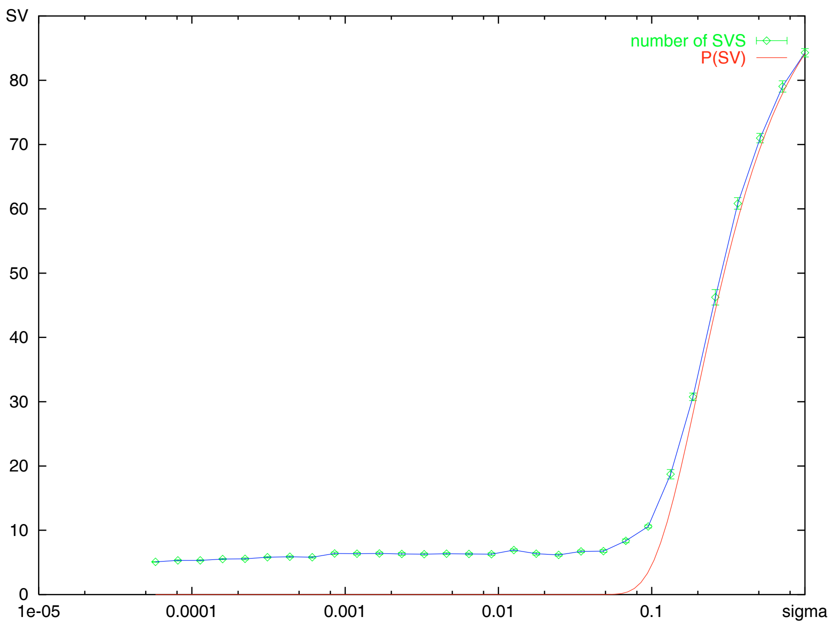


Figure 5.8: Number of Support Vectors for different noise levels and fixed $\epsilon$-margin of 0.2 (splines of degree 1, 100 points, 100 nodes, gaussian noise, 25 trials) and probability of a point to lie outside the margin for an unbiased estimator

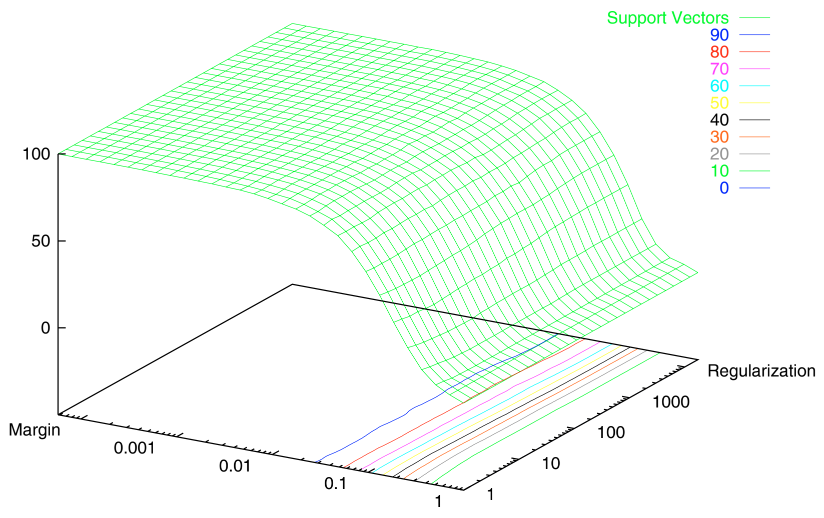


Figure 5.9: Number of Support Vectors depending on the regularization and the $\epsilon$-margin (splines of degree 1, 100 points, 100 nodes, gaussian noise with standard deviation 0.1, 26 trials)

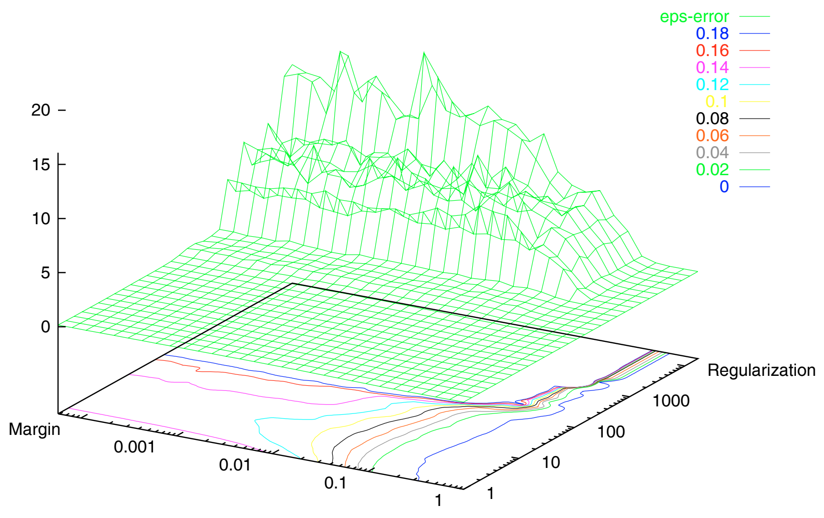


Figure 5.10: $\epsilon$-insensitive cost function depending on the regularization and the $\epsilon$-margin (splines of degree 1, 100 points, 100 nodes, gaussian noise with standard deviation 0.1, 26 trials)

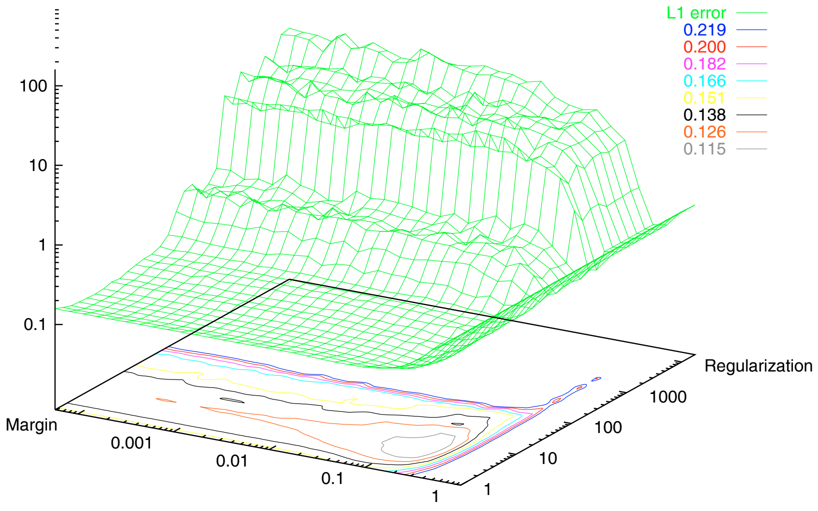


Figure 5.11: $L_1$-error of the approximation depending on the regularization and the $\epsilon$-margin (splines of degree 1, 100 points, 100 nodes, gaussian noise with standard deviation 0.1, 26 trials)

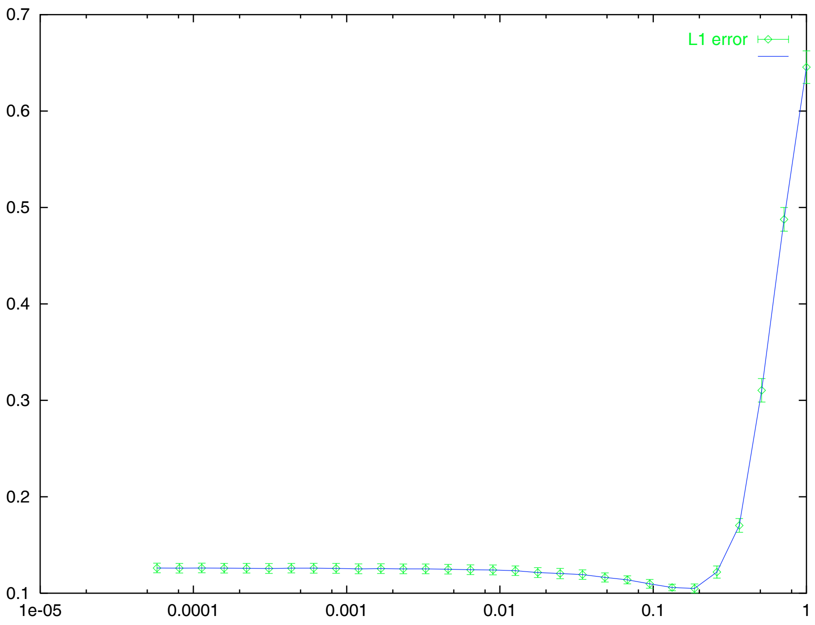


Figure 5.12: minimal $L_1$-error (optimal choice of the regularization) depending on the $\epsilon$-margin (splines of degree 1, 100 points, 100 nodes, gaussian noise with standard deviation 0.1, 26 trials)

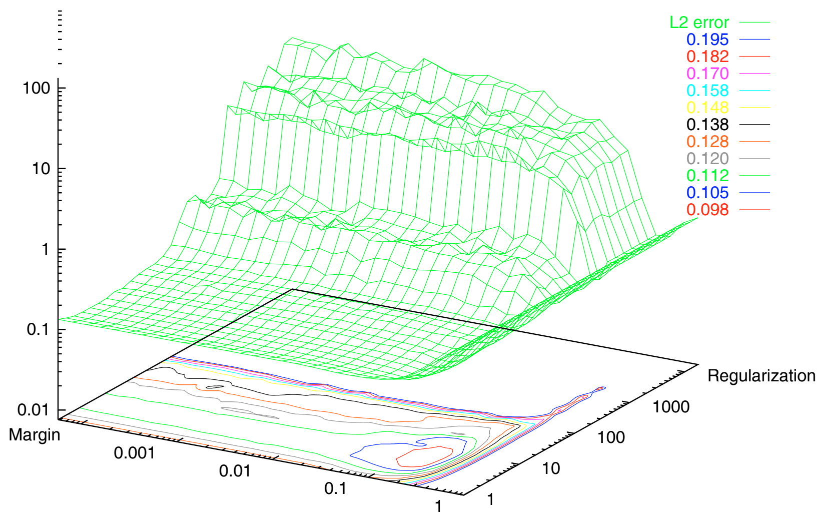


Figure 5.13: $L_2$-error of the approximation depending on the regularization and the $\epsilon$-margin (splines of degree 1, 100 points, 100 nodes, gaussian noise with standard deviation 0.1, 26 trials)

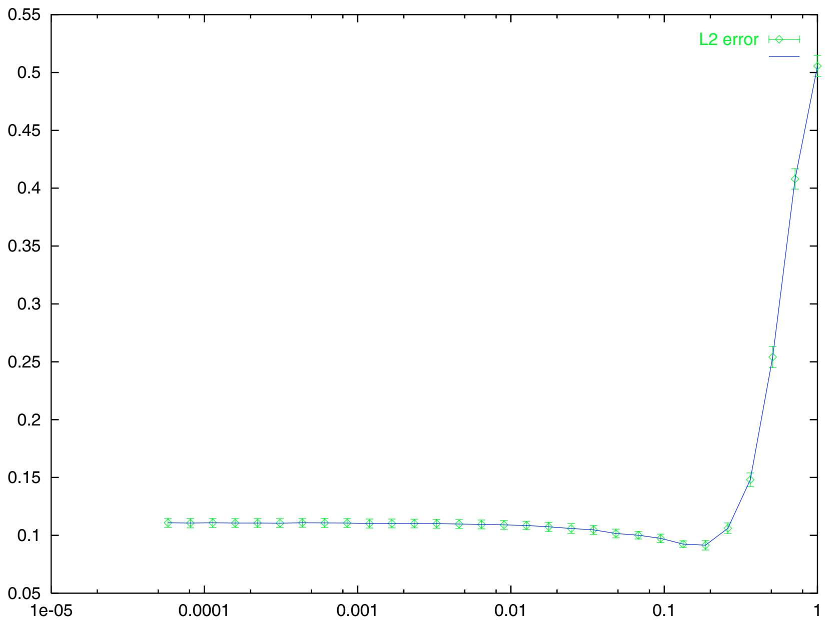


Figure 5.14: minimal $L_2$-error (optimal choice of the regularization) depending on the $\epsilon$-margin (splines of degree 1, 100 points, 100 nodes, gaussian noise with standard deviation 0.1, 26 trials)

Clearly for gaussian noise an $L_2$ loss function would be the best choice. Hence an $\epsilon$-insensitive loss function with nonzero $\epsilon$ is a better approximation of the $L_2$ loss than a $L_1$ loss is (see figure 5.16). For the realizable asymptotic case one can compute the optimal $\epsilon$ exactly depending on the variance of the gaussian noise model.

### 5.2.3 Asymptotic Efficiency

Although no statement can be made so far about the optimal choice of the $\epsilon$-margin for finite sample size still one can compute the asymptotic efficiency for loss functions (under the assumption of realizability). This one is given by [Vap79], [MYA94], [Mur96a]

$$e = \frac{1}{\mathrm{Var}(\phi)I} = \frac{1}{Q^{-1}GQ^{-1}I} \tag{5.5}$$

where $\phi$ are the parameters of the density model and

$$G := E(\Delta l_\theta(\eta))^2, \quad \text{and} \quad Q := E(\Delta\Delta l_\theta(\eta)) \tag{5.6}$$

In our case we are dealing with a one-parametrical model hence denoting

$$\phi(\xi) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\xi^2}{2\sigma^2}} \tag{5.7}$$

$$\Phi(\xi) = \frac{1}{2(1+\epsilon)} e^{-\left\{\begin{array}{ll} 0 & \text{if } |\xi| \le \epsilon \\ |\xi| - \epsilon & \text{otherwise}\end{array}\right.} \tag{5.8}$$

Then we get

$$\begin{aligned} I &= E\left( (\frac{d}{d\xi} \ln \phi(\xi))^2 \right) = E(\frac{\xi^2}{\sigma^4}) && (5.9)\\ &= \frac{1}{\sigma^2} \\ G &= E\left( (\frac{d}{d\xi} \ln \Phi(\xi))^2 \right) = 2\int_\epsilon^\infty \phi(\xi) d\xi && (5.10)\\ &= 1 - \mathrm{erf}(\frac{\epsilon}{\sigma}) \\ Q &= E\left( \frac{d^2}{d\xi^2} \ln \Phi(\xi) \right) = 2\ln\phi(\epsilon) && (5.11)\\ &= \sqrt{\frac{2}{\pi\sigma}} e^{-\frac{\epsilon^2}{2\sigma^2}} \end{aligned}$$

$$\tag{5.12}$$

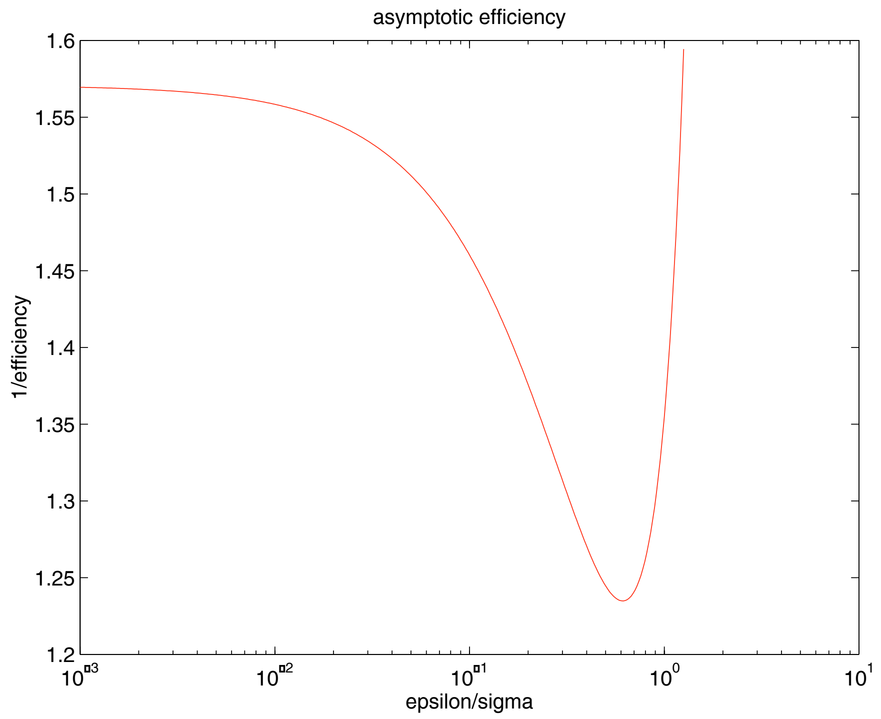


Figure 5.15: Asymptotic efficiency for an $\epsilon$-insensitive model and data with gaussian noise. See equation (5.13).

Now we can compute $e$

$$\frac{1}{e} = \frac{GI}{Q^2} = \frac{\pi}{2} e^{\frac{\epsilon^2}{\sigma^2}} \left(1 - \operatorname{erf}(\frac{\epsilon}{\sigma})\right) \tag{5.13}$$

Substituting $\tau := \frac{\epsilon}{\sigma}$ we get the maximum of $e^{-1}(\tau)$. This is obtained for $\tau = 0.6166$. It shows that there is a linear dependency between the optimal choice of $\epsilon$ and the noise of the data.

Figure 5.16 shows the relationship between a $L_2$ loss function and the corresponding optimal $\epsilon$-insensitive function. Note that the previous calculations are heavily based on the assumption of a *faithful* model and *infinitely* much data both of which is not true in our case. Therefore the calculus only gives a crude estimate of the $\epsilon$ that is optimal for finite sample size. Still it agrees well with the results obtained by other methods. For instance Solla & Levin [SL92b] prove that for Boltzmann machines (in the linear case) the best performance is achieved when the internal noise matches the external one.

This effect also can be observed in figure 5.17 for a fixed $\epsilon$-margin and varying noise (hence we're dealing with $e^{-1}(1/\sigma)$). Also notice that in the experimental finding the optimal performance is not obtained for $\tau = 0.6122$ but for $\tau \approx 0.9$. This may be true due to finite sample size effects and the fact that we're not in the *estimation of a location parameter* case but trying to infer a functional

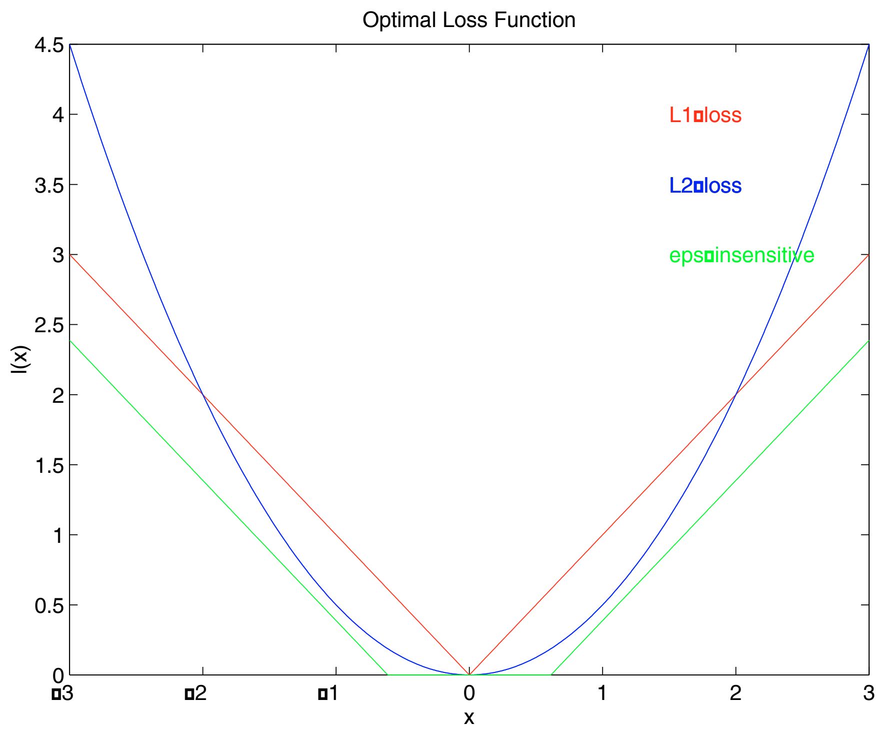


Figure 5.16: $L_1$, $L_2$ and $\epsilon$-insensitive loss functions in comparison. One can see that the $\epsilon$-insensitive approximates the $L_2$ function better than the $L_1$ does (piecewise linear fit with one more parameter).

dependency which is much more complicated than the original case.

Finally figures 5.18 and 5.19 show $L_1$ and $L_2$ loss functions for different choices of $\epsilon$ and $\sigma$. The minimum of these curves is shown in 5.20.

## 5.3 Future Perspectives

Still one question remains unanswered - the choice of the regularization constant $U$. Unfortunately the bounds which had been appplied for Pattern Recognition [GBV93] are not tight enough for Regression Estimation. One might consider a poor man's approach like cross validation or something more sophisticated like bootstrapping methods [ET94] until the problem is settled in a reasonable manner. This leaves a wide field of research open for future work.

Another aspect to be investigated is to find optimality criteria for approximating certain datasets, e.g. video, audio data .... Prior knowledge about the smoothness of the functions the algorithm is dealing with can be incorporated through the $n$-times differentiability of the kernels used.

Reduction of the complexity of the algorithm to linear scaling in the number of samples is an aspect to consider, too. A possible solution of this problem is

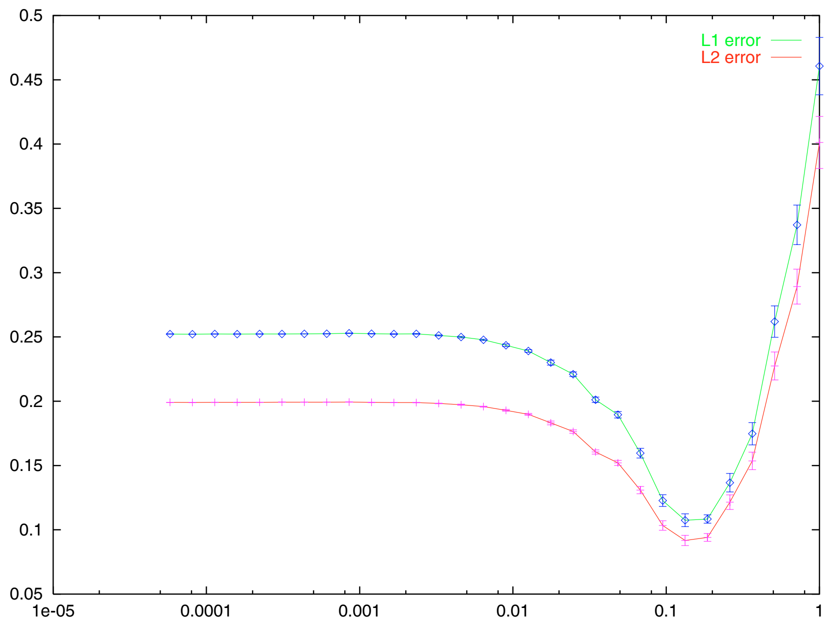


Figure 5.17: $L_1$ and $L_2$-error of the approximation for different noise levels and fixed $\epsilon$-margin of 0.2 (splines of degree 1, 100 points, 100 nodes, gaussian noise, 25 trials)

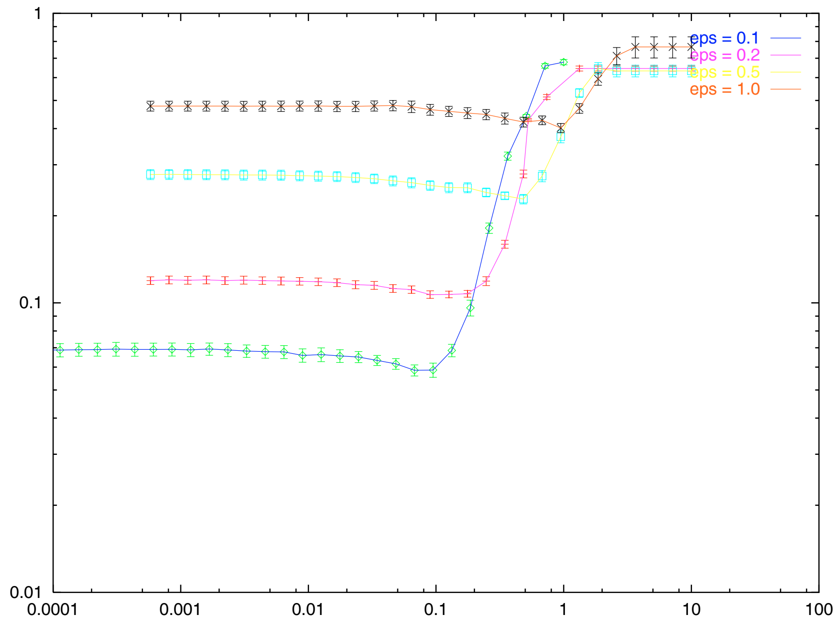


Figure 5.18: $L_1$-error for different noise levels

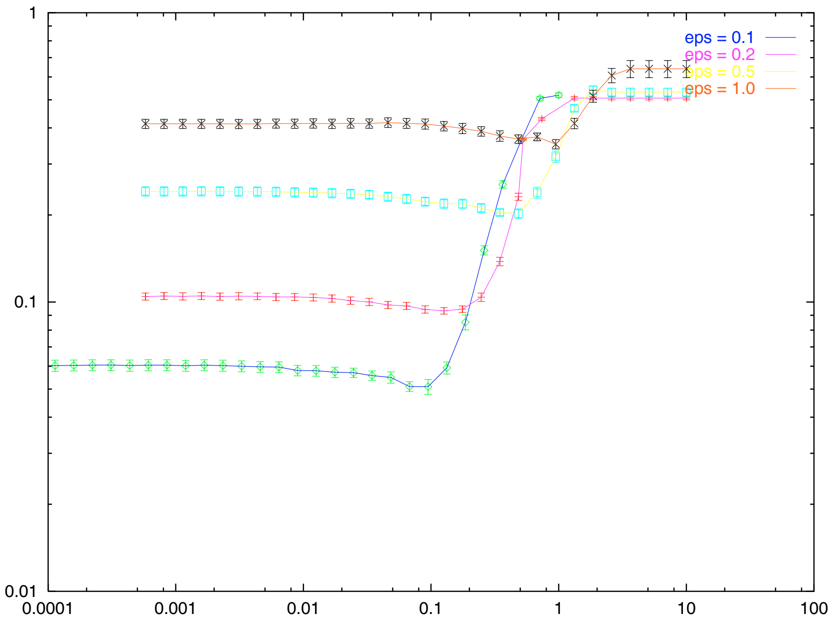


Figure 5.19: $L_2$-error for different noise levels

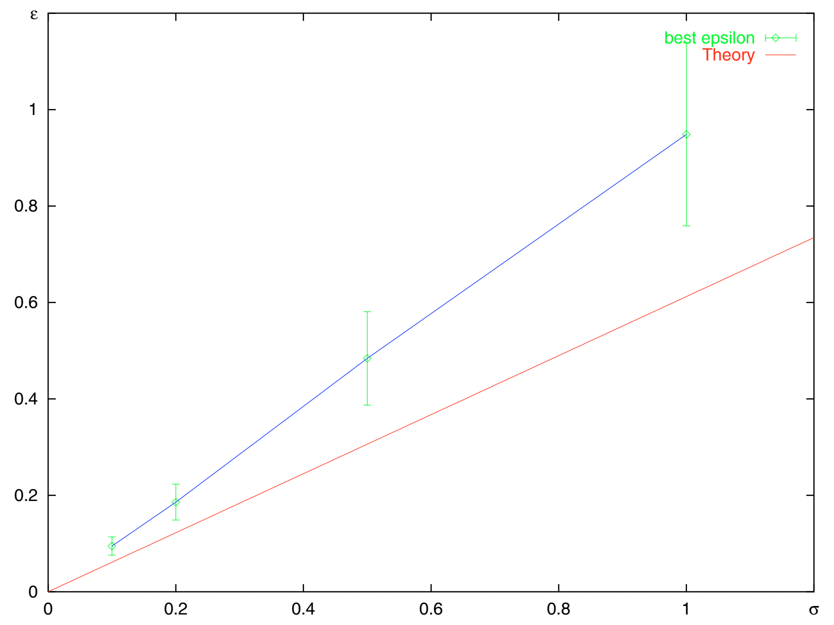


Figure 5.20: Optimal choice of $\epsilon$ for different noise levels (100 samples) and theoretically best value for the asymptotic unbiased case

solving the SV-equations locally only and updating the neighbouring domains with the error of the prior approximation. Compact support kernels might be very useful in this respect as they limit the *side effects* of a local solution to other domains.

# Chapter 6

# Summary

In this work we presented a method for approximating and estimating real functions $f : \mathbb{R}^n \longmapsto \mathbb{R}$ with virtually arbitrary sets of function expansions in a nonparametrical statistics setting.

Instead of exponentional computational effort in the number of input dimensions of the problem which is usually the case for classical spline–approximation methods the complexity of a Support Vector Machine depends only on the number of samples used. For data with regularities this means that computation time can be as small as $O(l^2)$ in the number of datapoints and $O(l^3)$ for the generic case. No special previsions have to be taken for the choice of a predefined grid or any similar measure necessary for conventional methods (finite elements, adaptive splines, etc.). This means that one can handle approximation/estimation problems very effectively.

Moreover we can deal with a large variety of loss functions in order to suit everyone's need when trying to estimate the best solution for a real–world problem. The loss functions do not even have to be known explicitly - knowledge is required only at the sampling points of the data.

For the $\epsilon$–insensitive loss function we can get even more interesting results as it entails a lot of coefficients $\alpha_i$ that vanish in the support vector decomposition. This can be regarded as a novel way of data compression. Still a lot of work will have to be spent on this issue to make it a practical way of encoding data.

We presented some theoretical connection between the choice of $\epsilon$ and the noise of the data although this result can be regarded as a preliminary one only. This is a hard problem indeed as it means estimating how well one may trust the data given. The second free parameter still remains to be chosen in a theoretically more sound way. Whereas in the pattern recognition case good bounds on the capacity of the set of functions could be obtained (thereby yielding an effective way of finding the correct regularization of the data) we still have to do the same for regression estimation. The bounds from Pattern Recognition are not tight enough. This leaves the Support Vector Regression Estimation as a powerful algorithm with still a lot of work to be done in the following years.

# Appendix A

# Implementation

## A.1 Choice of a programming language

Initially there was code available in a Lisp-like environment (SN3.1) with mixed compilation and interpreter features for rapid prototyping. Besides that a large set of libraries for matrices, display routines and general utility code was available. It had been tested and co-developed in the department which guaranteed a high degree of adaptation to the specific problems.

Nevertheless it had several shortcomings. The documentation was partly missing or obsolete which caused inefficiencies in coding. Further deficiencies were the awkward prefix syntax rendering the code very hard to check for errors and the only half way implemented object orientation of the language itself. Derived classes for instance could not explicitly call base class constructors thereby creating the need for several workarounds using macro expansions (which were even more unreadable).

Lastly the system was available for SUN-OS 4.1 only with several mutually incompatible versions of the environment. The number of worldwide installations could be considered rather small. Therefore we decided to switch to a more mainstream development environment.

At this point the decision was taken to choose ANSI-C++ as we considered its standardization to have advanced far enough and there was an award winning integrated development system by SunSoft available (SparcWorks) including a powerful source code debugger with interpreter-like features. The availability of a large number of class libraries for nearly any purpose was expected to cope with the initial lack of "homegrown" code.

ANSI-C would have provided a higher portability and better compilers due to the meanwhile completed standardization but was not powerful enough due to the lack of object orientation. Fortran was not taken into consideration because of its arcane syntax and Java due to the lack of standardization and still unsatisfactory numerical performance as interpreted bytecode.

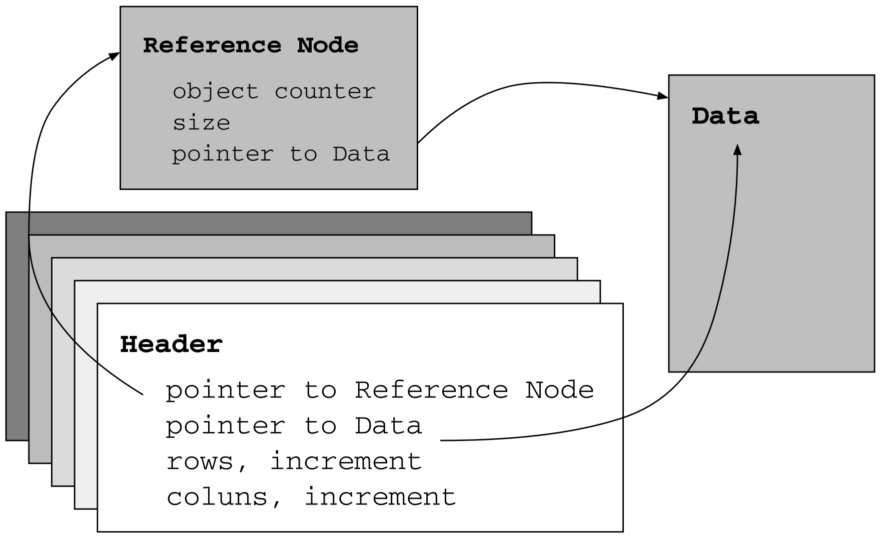


Figure A.1: Structure of a matrix / vector object

## A.2 Class Libraries

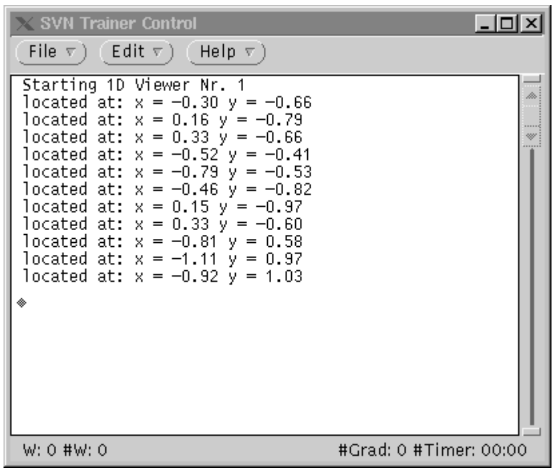
The basic ingredients for a Support Vector Learning Machine are a linear algebra package, a quadratic optimizer and tools for data visualization and entry.

### A.2.1 Linear Algebra Package

The matrix class was required to handle fast (pointer arithmetics) memory access with minimal overhead as well as non memory copying submatrix constructors. Moreover it was expected to handle large (approx. 200 MB) memory segments efficiently cooperating well with the operating systems memory management facilities. This library was regarded a key performance issue.

The currently available matrix classes were mainly lacking the feature of efficient direct pointer arithmetics, were not template based thereby allowing only a limited class of arithmetic types or provided just an overhead to old legacy Fortran code (blas, lapack and its c++ equivalents). Other libraries had the serious defect of not being available as source code (Rogue Wave) for eventually necessary extensions. The ANSI C++ ValArray Template which will be part of the Standard C++ Library seemed appealing to start with but was not available at the time the project started. Therefore we decided to write our own matrix class. It‘s basic structure can be seen in figure A.1.

Creating a submatrix requires only the creation of a new header referring to



```
Starting 1D Viewer Nr. 1
located at: x = -0.30 y = -0.66
located at: x = 0.16 y = -0.79
located at: x = 0.33 y = -0.66
located at: x = -0.52 y = -0.41
located at: x = -0.79 y = -0.53
located at: x = -0.46 y = -0.82
located at: x = 0.15 y = -0.97
located at: x = 0.33 y = -0.60
located at: x = -0.81 y = 0.58
located at: x = -1.11 y = 0.97
located at: x = -0.92 y = 1.03
```

```
W: 0 #W: 0                    #Grad: 0 #Timer: 00:00
```

Figure A.2: main control window

the storage object. To avoid unwanted deletions of the storage object a bookkeeping variable in the storage object counts the current number of headers created. As for library functions it provides fast (pointer arithmetic) dot products of all kinds (including triangular matrices), Cholesky decompositions, sorting, iterations and indexing routines.

### A.2.2 Data Visualization

Ease of use and cross platform portability were key issues in selecting a suitable windows library. Therefore it was clear not to use the X11 or Motif libraries directly. Cross platform environments like the SolarSystem by Star Division were overly expensive and not available as source code. SunSoft's Visual Workshop, an integrated GUI-Builder (Graphical User Interface) Kit which would have allowed true Unix to MS-Windows portability by embedding MFC-calls (Microsoft Foundation Class) in the toolkit appeared too lately to be integrated. Howewer there was a library available that met all the basic requirements - wxWindows by the AIAI (Artificial Intelligence Applications Institute) in Edingborough, written by Julian Smart. Source code availability, true cross platform portabiliby (Windows, OS/2, all flavours of Unix, MacOS 7), a large number of high level classes and a precise documentation, all covered by the GNU General Public License convinced us to choose this tool. It proved very valuable for rapid prototyping (cfr. figure A.2 and A.3).

Currently work is being done on generating a Web-Browser compatible Java Interface for enhanced portability and easier commercialization of the system.

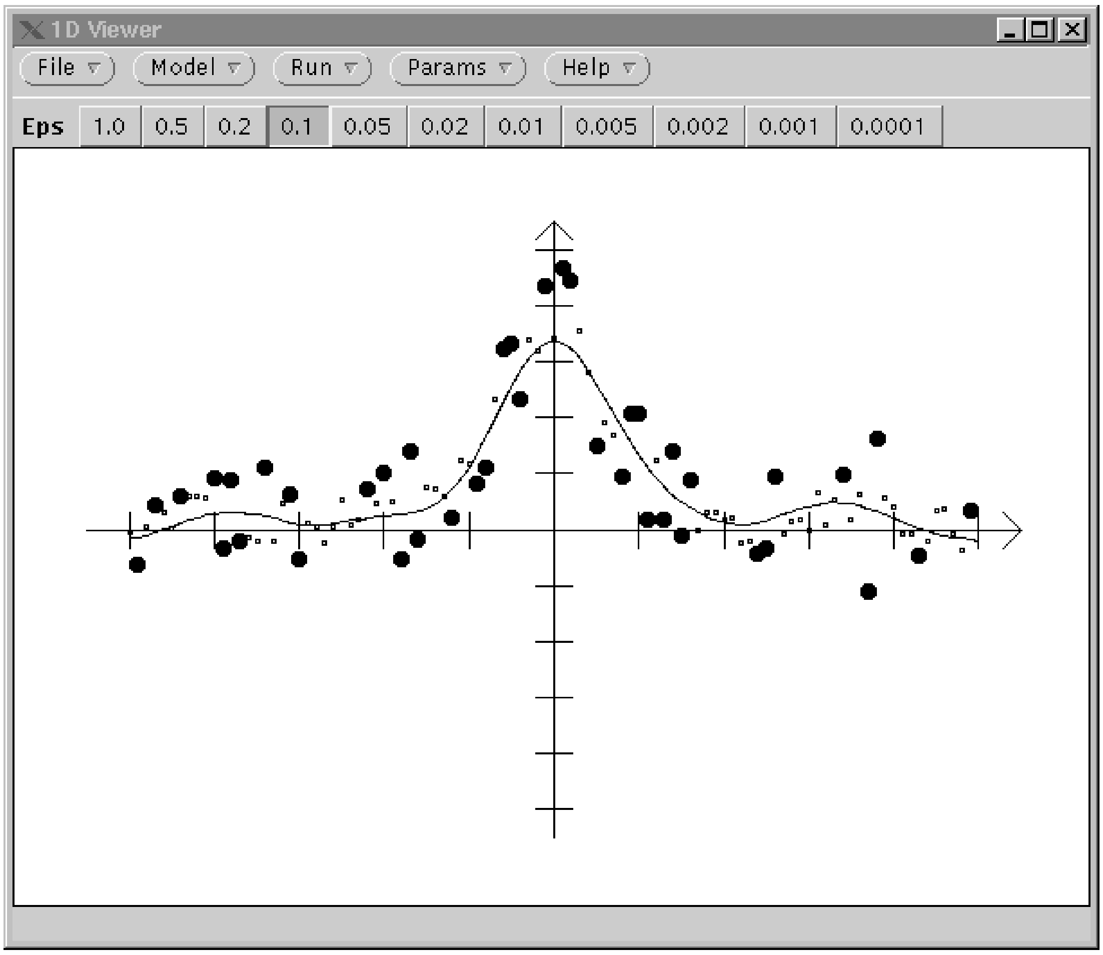


Figure A.3: a typical session for 1D-regression

### A.2.3 Optimization

There are several commercial packages available for quadratic programming like OSL by IBM (cfr. [IBM92]). It uses a two part algorithm to minimize a quadratic objective function with a positive semidefinite quadratic coefficient matrix subject to linear constraints. Since the optimum may occur in the interior of the feasible region, the simplex method alone cannot be used to solve QP problems. The first subalgorithm solves an approximating LP problem, using the simplex solver, and a related very simple QP problem at each iteration. When successive approximations are close enough together, the second subalgorithm, which permits a quadratic objective and converges very rapidly from a good starting value, is used. CPLEX (cfr. [CO94]) instead uses a primal-dual logarithmic barrier algorithm with predictor-corrector (cfr. [VDY94]).

Another package, MINOS by the Stanford Optimization Laboratory (cfr. [MS83]) uses a reduced gradient algorithm in conjunction with a quasi-Newton algorithm. The constraints are handled by an active set strategy. Feasibility is maintained throughout the process. The variables are classified as basic, superbasic, and nonbasic; at the solution, the basic and superbasic variables are away from their bounds. The null space is spanned by a matrix that is constructed from the coefficient matrix of the basic variables by using a sparse factorization. On the active constraint manifold, a quasi-Newton approximation to the reduced Hessian is maintained.

The basic ideas of LOQO (cfr. [Van94]) already have been explained in chapter 4. It uses a primal-dual interior point path-following method. Benchmarks on the Netlib problem collection showed superior performance over MINOS and CPLEX. Moreover it is freely available for academic research purposes. Hence we decided to use this package for implementation.

Another system by L. Kaufman (cfr. [BK80], [BK77], [DBK$^+$96]) uses an iterative free set method starting with all variables on the boundary and adding them as the Karush Kuhn Tucker conditions become more violated. This approach has the great advantage of not having to compute the full dot product matrix from the beginning. Instead it is evaluated on the fly, yielding a significant performance improvement.

In order to suit these demands we designed a very general optimization class. Calling parameters are a set of lagrange multipliers and a general type of optimization problem parameters like (4.1). The class also provides handles for dynamic calculation of the dot product table.

# Appendix B

# MPS Format

The MPS-Format is a pretty much arcane format for interfacing a system with an optimizer through an ASCII file. It stems from the times when FORTRAN was still very fashionable and is extremely sensitive to misplaced spacings and other whitespace. Nonetheless it is an industrial standard and therefore one has to deal with it (CPLEX, LOQO, MINOS and other LP/QP-solvers use it). Unfortunately information can be obtained only very difficultly. I will show a sample configuration for a simple constrained quadratic optimization problem.

## B.1 The Problem

$$\begin{aligned} \text{minimize } W(\vec{\alpha}) &= (\vec{\mathbf{t}}, \vec{\alpha}) + \frac{1}{2}\vec{\alpha}^t D\vec{\alpha} \qquad (B.1) \\ \text{subject to } \quad & (\vec{\mathbf{c}}, \vec{\alpha}) = 0 \\ & 0 \leq \alpha_i \leq U \quad \forall i \in \{1..l\} \end{aligned}$$

## B.2 The File

Beware, positions of the keywords and numbers are crucial! The column numbering is given for convenience and is *not* part of the file format itself. Labels are eight characters at most (case matters for some optimizers), numbers are right aligned and floating point without exponential notation which drastically limits the numerical resolution of the data entry (10 characters are available for numbers plus a trailing sign). Substitute the appropriate numbers for $t_i, c_i, D_{ij}$ in the sample file. Possible floating point numbers are: 4.235 3455. .235 $-235.3$ (note the '.' after the integer).

### B.2.1 Keywords

| | |
|---|---|
| NAME | name of the problem<br>(also specifies the output filename) |
| ROWS | linear part of the objective function and constraints<br>N no constraint specified (for objective function only)<br>G greater<br>E equal<br>L less |
| COLUMNS | contains the variables to be computed (here $\alpha$); two lines per row are allowed the labels specify the corresponding rows |
| RHS | right hand side of the ROWS section |
| BOUNDS | (optional) box constraints on the variables, default setting is $variable \geq 0$<br>LO lower bound<br>UP upper bound |
| QUADS | elements of the quadratic matrix, unspecified elements are set to 0 |

### B.2.2 Listing

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890123456789012345

NAME          svn.mps
ROWS
 N  objectiv
 E  constrai
COLUMNS
    alpha_01  objectiv            t1   c_000001            c1

    .         .                   .    .                   .
    .         .                   .    .                   .
    .         .                   .    .                   .

    alpha_0n  objectiv            tn   c_000001            cn
RHS
    rhs       constrai            0.
BOUNDS
 UP BOUND     alpha_01             U

 .  .         .                    .
 .  .         .                    .
 .  .         .                    .
 UP BOUND     alpha_0n              U
QMATRIX
    alpha_01  alpha_01            D11
    alpha_01  alpha_02            D12

    .         .                   .
    .         .                   .
    .         .                   .

    alpha_0n  alpha_0n            Dnn
ENDATA

12345678901234567890123456789012345678901234567890123456789012345
         1         2         3         4         5         6
```

# Bibliography

[ABR64] M. A. Aizerman, E. M. Braverman, and L. I. Rozonoér. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[Aka74] H. Akaike. A new look at the statistical model identification. *IEEE Trans. Automat. Control*, 19(6):716–723, 1974.

[BBdHS95] A. W. Bojanczyk, R. P. Brent, F. R. de Hoog, and D. R. Sweet. On the stability of the bareiss and related toeplitz factorization algorithms. *SIAM J. Matrix Anal. Appl.*, 16(1):40–57, January 1995.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[BGV92] Bernhard E. Boser, Isabell M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In David Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

[Bis95a] C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[Bis95b] C. M. Bishop. Training with noise is equivalent to Tikohonov regularization. *Neural Computation*, 7:108–116, 1995.

[BK77] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31:163–179, 1977.

[BK80] J. R. Bunch and L. Kaufman. A computational method for the indefinite quadratic programming problem. *Linear Algebra and Its Applications*, pages 341–370, December 1980.

[Bun85] James R. Bunch. Stability of methods for solving toeplitz systems of equations. *SIAM J. Sci. Stat. Comput.*, 6(2):349–364, 1985.

[CH53] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, Inc, New York, 1953.

[CO94] Inc. CPLEX Optimization. Using the CPLEX callable library. Manual, 1994.

[CV95] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[Cyb88] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, Massachusetts, 1988.

[Dau92] I. Daubechies. *Ten Lectures on Wavelets.* SIAM, Philadelphia, PA, 1992. Notes from the 1990 CBMS-NSF Conference on Wavelets and Applications at Lowell, MA.

[DBK$^+$96] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Linear support vector regression machines. In *Advances in Neural Information Processing Systems 9*, 1996. forthcoming.

[DH74] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1974.

[dH87] Frank de Hoog. A new algirithm for solving toeplitz systems of equations. *Linear Algebra and its Applications*, 88/89:123–138, 1987.

[ET94] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap.* Chapman and Hall, New York, 1994.

[Fle89] R. Fletcher. *Practical Methods of Optimization.* Jonh Wiley & Sons, New York, NY, 2 edition, 1989.

[Fri91] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, 1991.

[GBV93] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

[Gol86] H. Goldstein. *Classical Mechanics.* Addison-Wesley, Reading, MA, 1986.

[Hol96] Arne Hole. Vapnik-chervonenkis generalization bounds for real valued neural networks. *Neural Computation*, 8:6:1277–1299, 1996.

[Hub72] P. J. Huber. Robust statistics: a review. *Ann. Statist.*, 43:1041, 1972.

[IBM92] IBM Corporation. IBM optimization subroutine library guide and reference. *IBM Systems Journal*, 31, 1992. SC23-0519.

[KM95] Marek Karpinski and Angus Macintyre. Polynomial bounds for VC dimension of sigmoidal neural networks. Technical Report TR-95-001, International Computer Science Institute, Berkeley, CA, January 1995.

[LMS90] Lustig, Marsten, and Shanno. On implementing mehrotra's predictor-corrector interior point method for linear programming. Princeton Technical Report SOR 90-03., Dept. of Civil Engineering and Operations Research, Princeton University, 1990.

[MS83] B. A. Murtagh and M. A. Saunders. MINOS 5.1 user's guide. Technical Report SOL 83–20R, Stanford University, CA, USA, 1983. Revised 1987.

[MS92] S. Mehrotra and J. Sun. On the implementation of a (primal–dual) interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[MT91] Jorge J. More and Gerardo Toraldo. On the solution of large quadratic programming problems with bound counstraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[Mur96a] N. Murata. Personal communication, 1996.

[Mur96b] Noboru Murata. An integral representation of functions using three–layered networks and their approximation bounds. *Neural Networks*, 9(6):947–956, 1996.

[MYA94] N. Murata, S. Yoshizawa, and S. Amari. Network information criterion—determining the number of hidden units for artificial neural network models. *IEEE Transactions on Neural Networks*, 5:865–872, 1994.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.).* Cambridge University Press, Cambridge, 1992. ISBN 0-521-43108-5.

[SB93] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer Verlag, New York, second edition, 1993.

[SBV95] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining*, Menlo Park, CA, 1995. AAAI Press.

[Sch46] I. J. Schoenberg. Contribution to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Math.*, 4:45–99, 112–141, 1946.

[SHKT95] Charles J. Stone, Mark Hansen, Charles Kooperberg, and Young K. Truong. Polynomial splines and their tensor products in extended linear modeling. Technical report, University of California at Berkeley, Berkeley, California, January 1995.

[SL92a] J. Sjöberg and L. Ljung. Overtraining, regularization, and searching for minimum in neural networks. In *Proc. of the IFAC Symposium on Adaptive Systems in Control and Signal Processing*, pages 669–674. IFAC, 1992.

[SL92b] Sara A. Solla and Esther Levin. Learning in linear neural networks: The validity of the annealed approximation. *Physical Review A*, 46(4):2124–2130, August 1992.

[TA77] A. N. Tikhonov and V. Y. Arsenin. *Solution of Ill–Posed Problems.* Winston, Washington, DC, 1977.

[Tim60] A. F. Timan. *Theory of Approximation of Functions of a Real Variable.* Dover Publications Inc., New York, NY, 1960. reprinted 1994.

[UAE91] Michael Unser, Akram Aldroubi, and Murray Eden. Fast B-spline transforms for continuous image representation and interpolation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(3):277–285, March 1991.

[Van94] R. J. Vanderbei. LOQO: An interior point code for quadratic programming. Technical report, Program in Statistics & Operations Research, Princeton University, Princeton,NJ, 1994.

[Vap79] V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian].* Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, New York, 1995.

[VDY94] R. J. Vanderbei, A. Duarte, and B. Yang. An algorithmic and numerical comparison of several interior-point methods. Technical Report SOR-94-05, Program in Statistics and Operations Research, Princeton University, July 1994.

[VGS96] V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advances in Neural Information Processing Systems 9*, 1996. forthcoming.

[Wal94] Gilbert G. Walter. *Wavelets and Other Orthogonal Systems with Applications.* CRC Press, 1994.