Function Approximation for Learning Control

This thesis has been completed in partial fulfillment of the requirements of the Dutch Institute of Systems and Control (DISC) for graduate study.

The research described in this thesis has been conducted within the Drebbel Institute for Mechatronics at the University of Twente, and has been financially supported by the 'Duits-Nederlands Mechatronica Innovatie Centrum.'

ISBN 90-365-2050-9

FUNCTION APPROXIMATION FOR LEARNING CONTROL
— A KEY SAMPLE BASED APPROACH —

Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit Twente, op gezag van
de rector magnificus, prof. dr. F. A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen op
vrijdag 18 juni 2004
om 15.00 uur

door

Bastiaan Johannes de Kruif
geboren op 31 januari 1976
te Amsterdam

Dit proefschrift is goedgekeurd door:

Prof. dr. ir. J. van Amerongen, promotor
Dr. ir. T. J. A. de Vries, assistent-promotor

Voorwoord

ZO, DAAR IS IE DAN. Na iets meer dan vier jaar werk ligt het resultaat hiervan bij u op tafel. Omdat ik het hele proefschrift heb geschreven, verwacht ik dan ook van u als lezer, dat u het geheel gaat lezen. Echter, als u met minder plezier het stuk moet doorworstelen dan dat ik het heb geschreven, dan ben ik tevreden als u het voorwoord leest, zodat u in elk geval weet dat dit werk niet alleen door mij tot stand is gekomen, maar dat ik in het tot stand brengen van dit stuk door velen ben geholpen.

Allereerst wil ik mijn begeleiders Job en Theo van harte bedanken voor de tijd en moeite die ze hebben gestoken in mijn werk. Door hun hulp geloof ik dat dit proefschrift tot een leesbaar resultaat is gekomen.

Door de sfeer op de vakgroep heb ik met veel plezier gewerkt. Zonder alle collega's die de sfeer samen creëerden zouden de lunches, borrels en ping-pong wedstrijden een stuk minder leuk zijn geweest. Van mijn collega's wil ik kamergenoot Jan speciaal in het zonnetje zetten, voor alle nuttige zinvolle discussies en even zo nuttige zinloze discussies.

Een deel van het werk bestond uit het schrijven van computerprogramma's. Door de hulp van Frank en Paul zijn deze programma's gaan doen wat ik wilde dat ze zouden gaan doen, en zijn ze hopelijk ook nog voor anderen te begrijpen. Hiervoor, en voor de koekjes die bij jullie op de kamer liggen, bedankt. Ook de studenten die ik heb mogen begeleiden wil ik bedanken, voor hun eigen ideëen over het onderzoek, en in het tonen waar de knelpunten zaten.

Mijn ouders kan en wil ik voor zoveel bedanken, dat ik me beperk tot slechts één onderdeel dat direct invloed op het proefschrift heeft: het nakijken op spelfouten. Mocht u nog een spelfout vinden, dan hoeft u dat ook niet aan mij te vertellen, maar kunt u hun een email sturen.

Als laatste wil ik Sita bedanken, gewoon omdat ze zo lief is.

Samenvatting

IN DIT PROEFSCHRIFT WORDEN benadermethoden geïntroduceerd voor het gebruik in lerende regeltechniek. Benadermethoden zoeken een relatie tussen in- en uitgang aan de hand van aangeboden data-punten. Twee verschillende, maar sterk gerelateerde methoden passeren de revue, namelijk de 'key sample machine' en de 'recursive key sample machine.' De eerste methode is voor off-line benaderingen, waarbij alle data tegelijk wordt aangeboden. De tweede methode is voor on-line benaderingen, waarbij de data-punten puntsgewijs worden gepresenteerd. De benadermethoden bepalen zowel de structuur als de parameters van de relatie.

Beide benadermethoden representeren de relatie in de data met een (beperkt) aantal van de data-punten zelf, de zogenaamde sleutelpunten (Eng.: 'key samples'). Doordat de relatie wordt gevormd door een aantal data-punten, en dat *níet* noodzakelijkerwijs de ingangsruimte in kleinere ruimtes wordt opgesplitst, heeft het aantal ingangen weinig invloed op de kwaliteit van de benadering. Tevens is het door het toevoegen en verwijderen van sleutelpunten mogelijk om de structuur, dat wil zeggen de vorm, van de relatie aan te passen.

De benaderingsmethoden bevatten een selectiemechanisme dat sleutelpunten toevoegt, en daarmee het aantal te benaderen relaties uitbreidt, totdat het onwaarschijnlijk is dat deze verdere uitbreiding enige relevantie heeft. Om deze onwaarschijnlijkheid te bepalen, wordt de kwaliteit van de data gebruikt, waardoor een nauwkeurige benadering gevonden wordt als de data dit toelaat, en een grove benadering als de data dit dicteert. Door toepassen van dit selectiemechanisme is het niet waarschijnlijk dat ruis wordt benaderd.

Een uitgebreide evaluatie en een vergelijking met andere benadermethoden laat zien dat een goede benadering van de relatie wordt gevonden. Voor deze benadering zijn in het algemeen minder parameters nodig dan de andere methoden nodig hebben. De geringe rekentijd

maakt het mogelijk om de geïntroduceerde methoden te gebruiken in lerende regeltechniek.

Een set van experimenten is uitgevoerd om de methoden te testen voor het doel waarvoor ze zijn ontworpen. Een verbeterd 'Learning FeedForward Control' regelschema is gebruikt in de experimenten. De experimenten tonen aan dat de benadermethoden, zowel off-line als on-line, in staat zijn om de volgfout van een mechanische opstelling significant te verbeteren. De benadermethoden zijn met succes getest in de experimenten met negen ingangen voor de benadermethoden.

Summary

TWO FUNCTION APPROXIMATORS are introduced in this thesis for use in learning control. These function approximators identify a relation between input and output based on samples. Two different, but closely related function approximators are introduced: the key sample machine and the recursive key sample machine. The first arrives at an approximation by processing all the data as a batch, while the second approximator arrives at an approximation by processing one sample a time. Both methods alter the structure and the parameters of the relation.

The function approximators represent the relation in the data with a (limited) subset of the training samples, the key samples. As the relation is formed by a subset of the training samples, the input space is *not* necessarily divided into regions. Therefore, the dimension of the input space has little influence on the accuracy of the approximation. Furthermore, by adding or removing key samples, it is possible to alter the structure, i.e. the form, of the relation.

Key samples are included, and with them the number of possible relations is extended, until it is improbable that this extension has any relevance. The quality of the data is used for calculation of the relevance. As a result of this relevance test, the approximation is accurate if the quality of the data allows it, or the approximation is rough, if the quality of the data dictates this. The selection mechanism prevents the current realisation of the noise from being fitted.

An extensive evaluation and a comparison with other approximators shows that a good approximation is found. In general, fewer parameters are required than other methods need. The limited computational load makes the methods applicable for learning control.

A set of experiments is conducted to test the approximators for the purpose they are designed for. An improved Learning FeedForward Control scheme is used in these experiments. The experiments show

that the (recursive) key sample machine is able to significantly reduce the tracking error of a mechanical setup, off-line and on-line. The approximators were successfully tested with nine inputs.

Contents

Nomenclature

Abbreviations and Acronyms

ANN	Artificial Neural Network
BSN	B-Spline Network
FEL	Feedback Error Learning
GLS	Generalised Least Squares
i.i.d.	Independently and identically distributed
ILC	Iterative Learning Control
KKT	Karush-Kuhn-Tucker
KSM	Key Sample Machine
LFFC	Learning FeedForward Control
LS	Least Squares
LSSVM	Least Squares Support Vector Machine
LSSVM+	Least Squares Support Vector Machine with optimal pruning
LSSVM(+)	LSSVM with or without optimal pruning
LWPR	Locally Weighted Partial Regression p. 111
MAE	Maximum Absolute Error
MAP	Maximum A Posterior
ML	Maximum Likelihood
MLP	Multilayer perceptron
MSE	Mean Squared Error
OBD	Optimal Brain Damage
OBS	Optimal Brain Surgeon

OLS	Ordinary Least Squares	
PSD	Power Spectral Density	
RBF	Radial Base Function	p. 46
RBFN	Radial Base Function Network	
RKSM	Recursive Key Sample Machine	
RLS	Recursive Least Squares	
RMS	Root Mean Square	
SVM	Support Vector Machine	
ZPETC	Zero Phase Error Tracking Control	

Calligraphic

$\mathcal{L}$	Lagrangian function	p. 157
$\mathcal{L}_{\mathrm{p}}^{-1}$	Inverse Laplace transform	
$\mathcal{Z}$	z-transform	

Greek Letters

$\boldsymbol{\alpha}$	Vector containing Lagrangian multipliers	
χ_i^2	χ-squared distribution with i degrees of freedom	
ϵ	Insensitivity zone of ϵ-insensitive cost function	eq. 2.11
$\boldsymbol{\varepsilon}$	Vector containing noise samples	eq. 3.3
ε	Zero-mean Gaussian noise	p. 9
κ	Condition number of matrix	
λ	Regularisation parameter, $\lambda \geq 0$	eq. 2.15
ϕ	Value of new indicator function for new key sample	
σ	Noise standard deviation	
$\xi^{(*)}$	Slack variable used for SVM	p. 38
ζ	Significance level of hypothesis rejection	

Operators and Functions

$\star$	Convolution	
$\lvert \bullet \rvert_\epsilon$	ϵ-insensitive cost function	eq. 2.11
$\hat{\bullet}$	Approximation of $\bullet$	
$\propto$	Proportional to	
$\bar{\bullet}$	Updated version of $\bullet$	
$C(\bullet, \circ)$	Cost associated by selecting $\bullet$ instead of $\circ$	
D	User defined operator for regulating the approximation	eq. 2.15
$E(\bullet)$	Expectation of $\bullet$	
$f()$	Indicator function	eq. 3.1
$\tilde{f}()$	Dual indicator function	eq. 3.42
$k()$	Kernel function	p. 40
$[\bullet]_{ij}$	Element i, j of $\bullet$	
$\lvert\lvert \bullet \rvert\rvert_2$	Two-norm of vector $\bullet$	
$\#\bullet$	Number of $\bullet$	
$P(\bullet)$	Probability distribution of $\bullet$	
$P(\bullet\lvert\circ)$	Conditional probability distribution of $\bullet$ given $\circ$	
$R(\bullet)$	Risk, or expected cost, for $\bullet$	
$R_{\text{emp}}(\bullet)$	Emperical risk for $\bullet$	

Roman Letters

$\mathbf{1}_i$	Identity vector	
$\mathbf{b}$	Parameter vector	
C	Regularisation parameter used in SVM. $C \propto \frac{1}{\lambda}$	p. 38
$C(s)$	Transfer function of controller	
d	Disturbance signal due to state-dependent effects, acting at plant input	fig. 1.2
$\mathbf{\Delta b}$	Change in the parameter vector	
D_{init}	initial size of the created regions for LWPR	p. 113

$\mathbf{e}$	Vector containing residuals, $\mathbf{e} = \mathbf{X}\hat{\mathbf{b}} - \mathbf{y}$	
$\mathbf{f}$	*Row vector* with indicator functions: $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})]$	p. 33
$\tilde{\mathbf{f}}$	Dual indicator vector, similar to $\mathbf{f}$	
$\mathbf{H}$	Hessian	
$\mathbf{I}$	Identity matrix	
$L(s)$	Transfer function of learning filter	
m	Mass	
N	Number of samples	
n	Number of indicator functions	
O	Order of low-pass filter	
$P(s)$	Transfer function of plant	
p_{n}	Noise density function	
$Q(s)$	Transfer function of low-pass filter	
r	Reference signal	fig. 1.2
r_i	Reference signal(s) for motor number i	p. 130
r_z	Reference signal for position ($z = \mathrm{x}$), velocity ($z = \mathrm{v}$) or acceleration ($z = \mathrm{a}$)	p. 126
S^2	Extra sum of squares	
T	Sampling time	
u	Control signal, $u = u_{\mathrm{fb}} + u_{\mathrm{ff}}$	fig. 1.2
u_{e}	Control signal acting on input of plant, $u_{\mathrm{e}} = u - d$	fig. 1.8(a)
u_{fb}	Feedback control signal	fig. 1.2
u_{ff}	Feedforward control signal	fig. 1.2
$\mathbf{V}$	Covariance matrix	p. 76
$\mathbf{V}^{-1}$	Weight matrix	p. 76
$\mathbf{X}$	Indicator matrix	eq. 3.2
$\mathbf{x}$	Vector of inputs on which the function depends	fig. 2.1(a)
$\mathbf{x}_i$	i^{th} observation of the input vector to the supervisor, also known as the input	
$\mathbf{X}_{\mathrm{ks}}$	Indicator matrix with only key samples	eq. 4.16
$\mathbf{x}_{\mathrm{new}}$	Input for a sample for which the output should be predicted	

$\mathbf{x}_u$	Vector of inputs on which the function does not depend	fig. 2.1(a)
$\mathbf{y}$	Vector with target samples	eq. 3.2
y	Output of the supervisor	fig. 2.1(a)
$\hat{\mathbf{y}}$	Vector containing estimates of the targets	
y_i	i^{th} observation of the output of the supervisor, also known as the target	p. 17
$\mathbf{y}_{ks}$	Vector with targets of key samples	eq. 4.20
y_t	Noise free target value	fig. 2.2
$\mathbf{z}$	Vector containing values of previous training samples with new indicator function (*unknown*)	
z_ζ	Value of realisation to reject hypothesis with significance ζ	

One

Introduction

THE FUSION OF CONTROL engineering and mechanical engineering results in the mechatronic approach to system development. In mechatronics, a system is designed as a whole, and not in separate blocks. The result of this approach is the development of inexpensive, high-performing systems (Van Amerongen, 2003).

An example of this approach in consumer electronics is the deskjet printer (Furman, Pinkernell, and Elgee, 1997). In a deskjet printer, a printer head moves over the paper driven by a belt to deliver ink where it is required. The belt in its turn is driven by a motor. Since the printing of digital photos becomes more and more popular, the motion of the printer head has to be precise to get sharp print-outs. On the other hand, the prints should not cost too much, or the consumer is not willing to buy the printer.

To satisfy these contradicting requirements, the design of the movement of the printer head should be a symbiosis of mechanical construction and control engineering. In that way, a construction is made that can be controlled with the specified precision, without wasting material or development time.

The control of the printer head is based on comparing the actual position of the printer head with the position desired. The difference between these positions, the error, is used to correct the actual position. This is called feedback control. Feedback control reacts to the error, while this error in itself is unwanted. A feedforward controller can be added to this scheme that responds to changes in the motion desired. When the desired printing head position starts to change, a force can be applied by the motor that moves the belt, *before* an error is introduced. The force that is necessary to steer the printing head with the correct

response is determined by a model. This model calculates the motion of the printer head as a result of some motor force. Incorporating a feedforward controller into the control scheme decreases the tracking error (Åström and Wittenmark, 1997).

Instead of *increasing* the tracking performance of the plant by feedforward, the feedforward controller can be used to *keep* the same tracking performance, with a less costly construction. One can think of motors with a larger force ripple or pulleys that are not exactly concentric. This approach, for a linear motor, is followed in De Kruif and De Vries (2002a).

A model of the plant is necessary to construct a feedforward controller. This model can be based on the physical interaction of the elements, or based on measurements. For a good feedforward signal, the non-linearities of the plant should be incorporated into this model. Since all printers are different, the feedforward controller has to be redesigned for each printer. This is a time-consuming task, and is therefore undesired. Furthermore, the plant might change in the course of time, e.g. because the ink-level in the printer head drops or the printer heats up, which requires adaptations to the feedforward controller.

Instead of identifying the plant with its non-linearities beforehand and build a feedforward controller for it, it would be advantageous to identify the plant while in operation and adapt the feedforward signal accordingly. This approach is used in several learning control schemes (Arimoto, Kawamura, and Miyazaki, 1984; De Kruif and De Vries, 2001b; Han, Kim, and Ha, 1998; Horowitz, 1994; Kawato, Furukawa, and Suzuki, 1987; Longman, 2000; Moore, 1992; Nørgaard, Ravn, Poulsen, and Hansen, 2000; Velthuis, 2000). To some extent, these approaches can manage deviances between the real plant and the model and compensate for them while the plant is in operation.

1.1 Motivation

A well-known learning control method is Iterative Learning Control (ILC) (Arimoto et al., 1984; Gorinevsky and Vukovich, 2001; Gunnarsson and Norrlöf, 2001; Han et al., 1998; Longman, 1998, 2000; Moore, 1992; Verwoerd, De Vries, and Meinsma, 2002).

ILC calculates a feedforward signal for a repetitive task that starts each run in the same state. The ILC scheme is illustrated in figure 1.1. Due to the repetitiveness of the task, disturbances that originate from the plant, like friction in motors, will introduce the same error each run. A feedforward signal as a function of the task-time can be calculated to

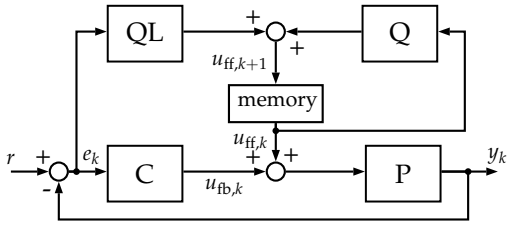
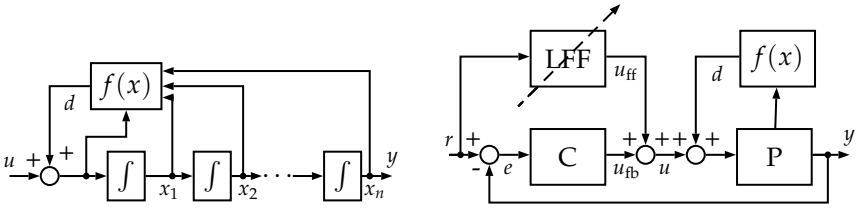


Figure 1.1: ILC scheme

compensate for these reoccurring errors. In this figure, C is a controller, P is a plant, L an (anti-causal) filter that calculates the feedforward signal to compensate for the remaining error and Q a low-pass filter to counteract modelling errors and noise. Returning to the printer example for this control scheme: if someone always prints the same photo of his wife, she is printed with higher precision after each try. This is because the tracking errors are reduced after each print by the adapting feedforward signal.

Because the feedforward signal is learnt as a function of the task-time, the ILC-scheme can only be applied if the task is repetitive. The feedforward signal learns that it has to apply some extra force at a certain time, but it does not relate this force to e.g. the friction. The Learning FeedForward Control (LFFC) scheme, however, learns the feedforward signal as a function of the states. Because the effects that require compensation also depend on these states, this feedforward signal can compensate for these effects, *independent of the motion* (De Vries, Velthuis, and Idema, 2001; Otten, De Vries, Van Amerongen, Rankers, and Gaal, 1997; Starrenburg, Van Luenen, Oelen, and Van Amerongen, 1996; Velthuis, 2000; Velthuis, De Vries, Vrielink, Wierda, and Borghuis, 1998). This may increase the print quality of all the photos, convenient if one also want to print birds. The effects the LFFC scheme compensates for, are *state dependent effects*; e.g. the LFFC scheme learns the friction as a function of the velocity and uses this relation to compensate for the error due to the friction. The assumed plant's model structure with the state dependent effects $f(x)$ is illustrated in figure 1.2(a). The forward path only contains integrators. The state dependent effects are assumed to be unknown. The purpose of LFFC is to approximate $f(x)$ as a function of x, the states of the plant, and then use it to cancel the signal, disturbing the input of the plant, d.

However, the states x are generally not available. To overcome this, LFFC bases its prediction on the reference signal and its derivatives, which makes it a feedforward controller, see figure 1.2(b). LFF is the



(a) Model of the plant

(b) The learning feedforward setting

Figure 1.2: LFFC setting and the model of the plant

learning feedforward controller containing the approximation of $f(x)$. The reference signals are easily accessible and noise free. The use of $f(r)$ instead of $f(x)$ assumes that their difference is small. If it would be large, the approximation cannot be used in the feedforward controller. This assumption holds if the state dependent effects are rather smooth and the motion desired is followed close enough by the feedback system alone. A complementary advantage of making it a feedforward scheme, is that the stability issue is less severe.

In order to approximate the state dependent function, some kind of function approximator is required. In previous work on LFFC, the used function approximator was a B-Spline Network (BSN) (Velthuis, 2000). This is a function approximator that consists of a set of basis functions of which only a small number has to be evaluated to calculate the output. The basis functions are active only in a specific region in the input space and do not contribute to the output of the approximator outside this region. The activation functions for ten B-spline functions are shown in figure 1.3. The activation of one of them is highlighted. The BSN will be treated in more depth in chapter 3. The output of a BSN is the sum of all the activation functions for the specific input, multiplied by their corresponding weight.

To divide the one-dimensional input space of figure 1.3, ten B-splines are used. If a two-dimensional space has to be covered with a similar distribution, $10 \times 10 = 100$ splines are required. The number of splines, and therefore the number of weights, grows exponentially with the input dimension. This results in large memory requirements, difficult training and a bad generalisation, and is known as the *curse of dimensionality* (Brown and Harris, 1994; De Vries et al., 2001). Due to this curse, the number of inputs is in practice limited to circa two. So, when the state dependent effects depend on more than two states, compensation by a BSN-based method become a problem.

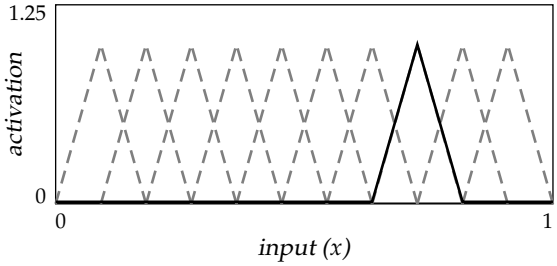


Figure 1.3: A set of B-splines on the input space

Alternative function approximators can be used to find the relation in a learning control scheme such as LFFC. Numerous function approximators can be found in the field of statistical learning theory (Vapnik, 1999, 2000), regression (Draper and Smith, 1998; Kleinbaum, Kupper, and Muller, 1987) and neural networks (Haykin, 1994; Zurada, 1992). However, these methods have not been designed to be embedded in a control system and hence might have characteristics that are unfavourable in that setting.

1.2 Problem definition

Because the function approximator is a part of the learning controller, the properties of the approximator will show in the behaviour of the learning controller; a learning controller with a BSN cannot be used if the effect that should be compensated for, depends on many inputs. Whenever this approximator will be used in another learning scheme, the same problem will arise. To avoid undesired behaviour of the complete setting, the characteristics of the function approximator should comply with the control scheme. This motivates the following problem definition:

Problem definition:

> *Find a function approximator that can be used in a learning control setting.*

Before a function approximator is looked for that can be used in a learning control setting, one should specify what conditions the approximator should fulfil so that it can be used. This will be the first sub-problem that has to be solved.

Sub-problem 1:

> *Determine the conditions for a function approximator to be applicable in a learning control setting.*

With the resulting set of conditions, a function approximator can be tested as to its applicability in a learning control setting.

Two situations are investigated in this thesis; in the first situation, all the data is present in a batch of samples. These samples are obtained in a separate training phase. In the second situation the data is present as a stream of samples. After each sample, the approximation needs to be updated. Because of these two situations, the problem definition is split into two parts:

Sub-problem 2:

> *Find a function approximator, or two separate function approximators, that comply with the conditions specified for a learning control setting and that can handle:*
>
> *2a) a batch of data,*
> *2b) a stream of data.*

Although the function approximator is developed for a general learning control setting, the LFFC case is the only case taken into consideration. This is done because the LFFC would exhibit problems if the function approximator did not function as required, just as other learning schemes would do. Furthermore, the LFFC setting calculates a feedforward signal for the current reference signal by means of this approximator. Because the feedforward controller is located in the feedforward path, instability can only occur if the transfer function of the LFFC becomes unbounded. This can easily be checked. So, from a stability point of view, LFFC is an attractive scheme. The last reason is that the method has been shown to work well in real-life setups (De Vries et al., 2001; Otten et al., 1997; Starrenburg et al., 1996; Velthuis, 2000; Velthuis et al., 1998).

1.3 Application

The applicability of the function approximator in a LFFC setting is tested on a Tripod, a pick-and-place machine. A photo and a schematic view of this setup are given in figure 1.4. This setup consists of three linear motors that can move up and down. A pair of rods is connected to each linear motor, and the other side of these rods to the platform at the top. Due to the constrained movements of the rods, the platform

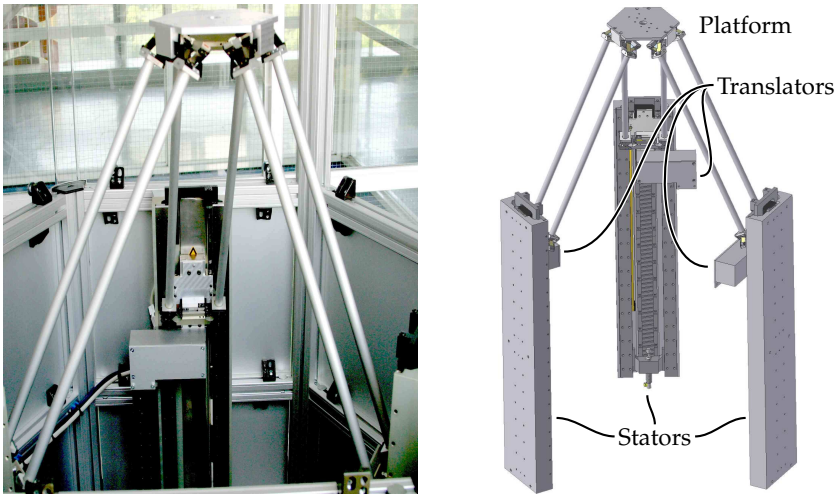


Figure 1.4: The Tripod

(a) Left position (b) Straight position

Figure 1.5: Possible parallelograms due to mechanical constraints

cannot rotate but only translate. The constraints on the rods make the rods form a parallelogram. This is illustrated in figure 1.5. The position of the platform is determined by the positions of the three linear motors. Only the position of the three linear motors is measured.

The Tripod is incorporated in a LFFC setting, see figure 1.2(b). The reference generator generates nine signals: position, velocity and acceleration for each of the three motors. The three forces to reduce the position error are calculated by the feedback controller. A feedforward signal is added to each of these three forces. These three feedforward

signals are calculated and learnt independently of each other. Each feedforward signal is calculated on the basis of (a subset of) the nine reference signals. After learning, the motors should be mainly controlled by the feedforward controllers. The outputs of the Tripod are the positions of the three motors. The feedforward signal is learned from the filtered error signal. This will be further dealt with in chapter 5.

The function approximator used in a learning control setting is tested on the Tripod. This application is chosen because of the following properties:

Motor characteristics: The motion of each individual linear motor is influenced by friction and cogging (Gieras and Piech, 2000). These state dependent effects act on the input of the motor that generates these effects; the cogging of motor number one will only influence motor number one. The cogging and friction of the motors are observable and can be compensated for by a learning feedforward controller. Apart from these effects, the (unknown) mass can be compensated for. Strictly speaking, the unknown mass is not a state dependent effect, because it acts as a multiplication factor in the forward path of figure 1.2(a). However, if the rate of the first integrator is included as 'state', so $\dot{x}_1$ of the model figure 1.2(a), the multiplication factor can be interpreted as a state dependent effect. This factor is determined by the algebraic loop form $\dot{x}_1$ to $f(x)$ and back again. The inclusion of the first rate as 'state' is maintained throughout this thesis.

The presence of these effects makes it attractive to use a learning controller.

Dynamic coupling: The motion of each motor will exert a force on the other motors due to the coupling via the platform. The effect of this force on a motor, can only be compensated for by the corresponding feedforward controller, if the motions of the other motors are known. This makes it necessary to have a feedforward controller with a high input dimension. The high input dimension was the bottleneck in previous research and with this coupling, it is tested if an approximator can handle high-dimensional input spaces.

Flexible number of inputs: The learning feedforward controller for each motor requires the reference signals of the other motors as inputs to compensate for the dynamic coupling. If a motor does not make a motion, it will not influence the other motors and therefore the reference signals of this motor are not required in the learning feedforward controllers of the other two motors. Consequently, these

controllers require only six inputs instead of the complete set of nine inputs. If two motors stand still, the learning feedforward controller only has to know the motion of its corresponding motor, because there is no dynamic coupling of the other motors. This results in three inputs.

So, by allowing one, two or three motors to move, the feedforward controllers require three, six or nine inputs. This characteristic can be used to test the sensitivity of the approximator to the number of inputs.

Limited stiffness: The stiffness of the construction is limited. Although the effect of limited stiffness on the learning controller is not a topic of research in this thesis, it will always be there, and it is therefore interesting to see its effect.

1.4 Data sets

Examples are included throughout this thesis to exemplify the ideas of the function approximators. Because function approximation concerns itself with finding a relation within a set of samples, two data sets containing samples are constructed to be used.

Data set 1 (Computer generated)
The first data set is generated by a computer, using:

$$y = \sin\left(\frac{1}{x+\nu}\right) + \varepsilon. \tag{1.1}$$

This function, with $\varepsilon = 0$, is shown in figure 1.6. The value of ν is set equal to 0.05 unless stated otherwise. x is uniformly distributed between zero and one. ε denotes noise with a zero-mean Gaussian distribution. The variance of this noise, as well as the number of training samples contained in the data set, will be changed throughout the thesis and will be given when used. A validation set is constructed by the same equation.

This function is used to generate data because of the difference in the absolute value of its derivative. It starts fluctuating fast while for larger values of x the fluctuation becomes very slow. This makes it possible to test whether the function approximator can handle different fluctuation speeds or not. This is furthermore interesting because the approximator should approximate the fast fluctuation function, while it should not fit the current realisation of the noise. Although this example is computer generated, it is connected to real-life applications, because in real-life it

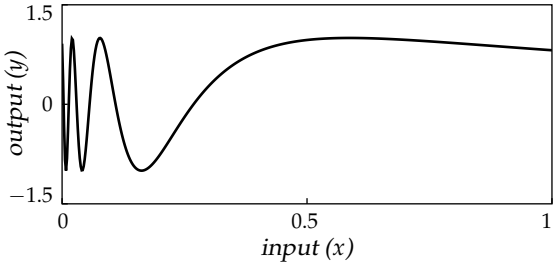


Figure 1.6: True function underlying data for data set 1

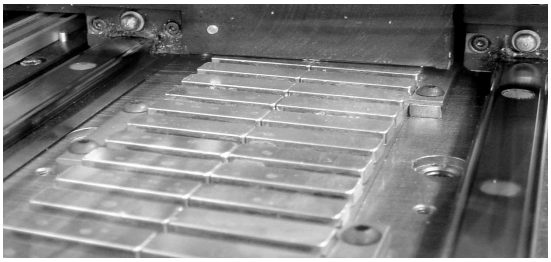


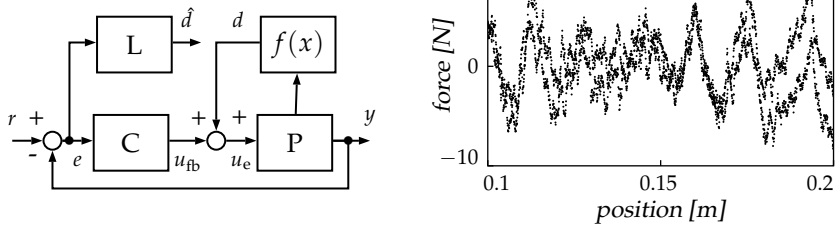
Figure 1.7: Magnets on the stator of a linear motor

is unknown whether, where and how wildly the function underlying the samples fluctuates. ▩

Data set 2 (Cogging force)
The second data set represents the cogging force as a function of the position, present in a linear motor. These measurements are made at a motor available at the Control Engineering laboratory of the University of Twente. The function approximation has to estimate the cogging force based on the position.

Cogging is caused by the attraction between the permanent magnets of the fixed part and the iron in the coils of the moving part (Gieras and Piech, 2000). Because the permanent magnets are placed periodically, the cogging force is periodic. A photo of the magnets placed on the stator of the linear motor is shown in figure 1.7. The data set of samples which relates the cogging force to the position, is obtained by filtering the error signal of the controlled linear motor. See figure 1.8(a).

Samples obtained in this way and which are used to estimate the relation, are shown in figure 1.8(b). The translator moved forth and back again to obtain these samples. The samples representing the cogging



(a) Measuring the cogging force

(b) Data set of measurements of the cogging force

Figure 1.8: Estimation of the cogging force with the values obtained

force are ambiguous, as can be seen from the position at 0.19 [m]. This ambiguous data is *not* removed by means of data pre-processing, because in a learning control setting, something alike may occur. The different approximators can be evaluated with respect to this set on how well they handle unknown noise levels and ambiguities in the data.

This data set contains 20 000 data samples, 2 000 of which will be used for validation purposes. ▩

1.5 Outline

In order to develop a function approximator that can handle a batch of data as well a continuous data stream, this thesis is structured as follows:

Chapter 2, Background: Before a review of function approximators can be made, or before an approximator can be constructed with the specific goal of usage in control, some background information has to be given. First, the setting will be dealt with from which the function approximator obtains its training data. This will cause the approximation problem to be formulated as optimisation problem. Different variants of this optimisation problem, and techniques to solve the problems, will be discussed.

The conditions put to the function approximator are treated in this chapter, so that in the following chapters these conditions can be used for evaluation purposes.

Chapter 3, Off-line function approximation: This chapter will treat the situation in which the data is present as a batch. A set of off-line function approximators will be investigated and evaluated. Based on this evaluation, the desired properties are combined to form a function approximator specifically for our goals. The performance of these approximators will be compared on the data sets noted before.

Chapter 4, On-line function approximation: An on-line function approximator is constructed based on the outcome of the off-line function approximator. A function approximator is described that is capable of handling a stream of data while it can still adapt its structure. Previously presented information is not omitted when the structure is adapted.

The approximator is evaluated to test its properties. Also a comparison with another approximator that alters its own structure online is made.

Chapter 5, Experiments: In this chapter the off- and on-line function approximator are tested in a phase-corrected Learning FeedForward Control setting. The control setting and practical considerations of the function approximator are dealt with. The experiments are done with the Tripod as plant in a learning control scheme. The results are shown for different experiments.

Chapter 6, Discussion: The last chapter will review the ideas presented in this thesis and relate these to the problem definition. Furthermore, some directions for future work are hinted at.

Two

Background

BEFORE A REVIEW of function approximators can be made, or before an approximator can be constructed especially for learning control, some background has to be given. This chapter treats the background of function approximation, and additionally, it will serve as a connection between the actual problem and the theory in subsequent chapters.

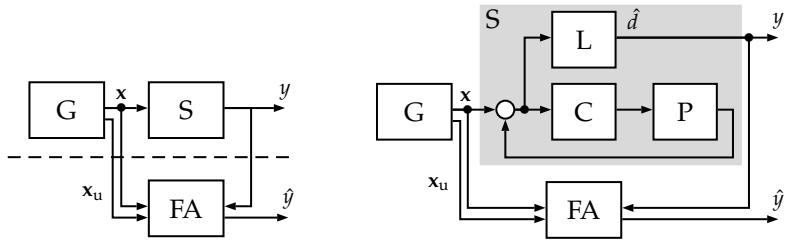
In order to make the problem clear, we start with the setting in which the function approximator has to operate. This setting indicates how the training data is generated. Based on this setting, a minimisation problem is formulated for the function approximator.

Different function approximators use different techniques to approximate the data. The characterisation of the different methods will be the subject of section 2.2, so as to create a handle by which these can be discussed. To treat the specific properties that are required for an approximator to be applicable in a control setting, section 2.3 is included.

2.1 Approximation problem

2.1.1 Approximation setting

In figure 1.8(b) samples of the cogging force are plotted as a function of the position. It is known from physics, that the cogging force depends on the position. When the position is not available for the prediction of the cogging force, no sensible prediction can be made. If a learning controller has to counteract some effect, it is important that the correct inputs are selected so as to predict the behaviour of the effect. However, the selection of the inputs is not within the scope of this thesis, and to



(a) The learning problem, based on figure 1.1 of (Vapnik, 2000)

(b) Obtaining samples to approximate the cogging set in the form of (a)

Figure 2.1: Learning setting, see also table 2.1

avoid situations in which inputs are missing, the following assumption is made.

Assumption 1 (All inputs present) *For a given approximation problem, the following holds:*

1a) The function approximator is supplied with all the relevant inputs.

1b) In addition to these inputs, irrelevant inputs might be present.

The learning scheme that is associated with this assumption, is shown in figure 2.1(a), which is based on figure 1.1 of (Vapnik, 2000). The upper part of the figure is the sample generating part, while the lower part of the figure is the function approximating part.

In this figure, G is a generator that generates vectors $\mathbf{x}$ and $\mathbf{x}_u$. Of these variables, only $\mathbf{x}$ acts as an input to the supervisor. The supervisor, S, assigns a value y to each input $\mathbf{x}$. The output of the supervisor is independent of the variable $\mathbf{x}_u$. However, $\mathbf{x}_u$ *is* presented as input to the function approximator, FA. This variable is included into the scheme, because it is assumed that irrelevant inputs can be presented to the function approximator. Apart from the irrelevant inputs, the function approximator is supplied with $\mathbf{x}$, which contains all the relevant inputs. The function approximator has to select the best function from a set of possible functions to predict the behaviour of the supervisor. After the selection, the function approximator can be used to predict the behaviour of the supervisor for inputs unencountered. The approximation of the true response y is indicated by $\hat{y}$. y is often referred to as the target for learning and, without loss of generality, it is assumed that y is a scalar.

Table 2.1: Relation between elements of the problem setting in figure 2.1(b) and figure 1.8(b)

Symbol	Description
S	controlled linear motor including data processing that calculates the cogging force
G	path generator
$\mathbf{x}$	position desired
$\mathbf{x}_u$	velocity and acceleration desired
y	estimate of cogging force, $\hat{d}$
$\hat{y}$	prediction of cogging force

Example 2.1
In figure 2.1(b) the scheme that was used to obtain samples of the cogging force is shown in the learning setting of figure 2.1(a). The relation between the variables used in the learning setting and the variables used in the setting of figure 1.8(b) are given in table 2.1.

The path generator R, generates samples for position, velocity and acceleration which the controlled linear motor should follow. However, the only variable that is necessary to predict the cogging is the position. So, this position is indicated by $\mathbf{x}$ in the learning setting. The cogging should be predicted and is therefore the target y. The true cogging force is unmeasurable, so the estimated cogging, $\hat{d}$ of figure 1.8(b) is used instead as the target. If other variables are fed to the function approximator as input, e.g. velocity and acceleration, which are not necessary for the prediction of the cogging, then these are contained in $\mathbf{x}_u$. ▩

An important aspect concerns the probability distribution by which the generator generates the samples. E.g. some areas of the state space of the Tripod are more likely to be visited than other areas; the middle part of the stators of the motors are more likely to be visited than the outer ends. To incorporate this, $\mathbf{x}$ and $\mathbf{x}_u$ are generated with a probability distribution $P(\mathbf{x})$ and $P(\mathbf{x}_u)$ respectively. A similar construction is used for the output of the supervisor, because uncertainties like noise corrupt it. The output of the supervisor is modelled as a conditional probability distribution depending on its input: $P(y|\mathbf{x})$. The distributions are assumed to be independently and identically distributed (i.i.d.).

The description of the conditional probability is rather broad and for reasons of simplicity, it is assumed that the noise is additive. The learning setting including this assumption is shown in figure 2.2. The

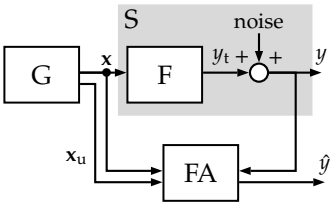


Figure 2.2: The learning problem with additive noise

new block name F is used to indicate that this represents a deterministic function with true output y_t. This noise-free output cannot be measured.

Assumption 2 (Additive noise) *The noise that acts on the true output y_t, is additive, independent of previous noise samples and is always generated by the same distribution.*

Throughout this thesis the terms function approximation and learning mechanism are used interchangeable. The function selected by these mechanisms is called the approximation or the estimate.

2.1.2 Minimisation problem

The goal of the function approximator is to select the 'best' function out of a set of functions to approximate the true response of the supervisor. In order to select the best function, a measure should be available to indicate whether one function is better than an other. Based on the Bayesian tradition, the measure is chosen to be the *risk*. The risk is defined as the expected cost:

$$R(\mathbf{b}) = \int_{-\infty}^{\infty} C(\hat{y}(\mathbf{x};\mathbf{b}), y)\ \mathrm{d}P(\mathbf{x}, \mathbf{x}_u, y). \tag{2.1}$$

In this equation $\mathbf{b}$ is a parameter vector that is tuned to minimise the expected cost. The cost function, $C(\hat{y}(\mathbf{x};\mathbf{b}), y)$, indicates the cost that is associated by selecting $\hat{y}$ instead of y. If the real economic cost is known, this cost can be used. However, this cost is often unknown and in the next section the question how the cost function can be chosen, based on the noise distribution, is answered.

The expectation of the cost is taken over the probability of the occurrence of a training sample, $P(\mathbf{x}, \mathbf{x}_u, y)$. Therefore, occurrences more likely to happen are predicted better. This is a useful property for learning control, because it means that the learning mechanism will learn better in areas more often visited. But, the probability distribution $P(\mathbf{x}, \mathbf{x}_u, y)$

is rarely known, and when it is not known, the risk cannot be minimised. To overcome this problem, the risk can be replaced by the *empirical risk*, i.e. the sum of the cost for all the samples in a *data set:*

$$R_{\text{emp}}(\mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N} C(\hat{y}(\mathbf{x}_i;\mathbf{b}), y_i). \tag{2.2}$$

In this, N is the number of samples, y_i the i^{th} output sample of the supervisor and $\mathbf{x}_i$ the i^{th} input sample to the supervisor. The prediction of the supervisor's behaviour comes down to finding the function that best approximates the outputs based on the inputs in the collection of samples called the data set. The integral over the combined probability in (2.1) can be approximated by the summation of the samples in (2.2), because a high probability is likely to result in more occurrences of samples in that area. Conditions for the set of functions so that the minimisation of the empirical risk converges towards the minimisation of the actual risk are given in Vapnik (1999, 2000) and can be summarised for the purposes of this thesis as: the set of functions considered in the function approximation problem should be bounded.

2.1.3 Maximum Likelihood

The cost $C(\hat{y}(\mathbf{x}_i;\mathbf{b}), y_i)$ used in the (empirical) risk has not yet been specified. Fisher (1912) links the additive noise distribution to the cost function, so that the likelihood of the given observation is maximised. This method is called the Maximum Likelihood (ML).

Maintaining the notation for the noise free output y_{t} of figure 2.2 and introducing $y_{\text{t},i}$ for the i^{th} observation of this variable, the likelihood of observation y_i is given as the conditional probability density function $p(y_i|y_{\text{t},i})$. Note that in the likelihood function $p(y_i|y_{\text{t},i})$, y_i is known and $y_{\text{t},i}$ is the variable, whereas this is just the other way around as one might expect.

The maximum likelihood estimator tries to maximise the expression $p(y_i|y_{\text{t},i})$ by varying $y_{\text{t},i}$, thus looking for the most probable value for a given set of observation. This can be written as:

$$\hat{y}_i = \arg\max_{y_{\text{t},i}} p(y_i|y_{\text{t},i})\,. \tag{2.3}$$

For a set of given observations the maximum likelihood is given as

$$\hat{\mathbf{y}} = \arg\max_{y_{\text{t},1}\cdots y_{\text{t},N}} \prod_{i=1}^{N} p(y_i|y_{\text{t},i}). \tag{2.4}$$

The value for which the argument of the maximisation problem is maximal, does not change by first applying the logarithm, because the logarithm is a monotonously increasing function. However, it will change the product of (2.4) into a summation:

$$\hat{\mathbf{y}} = \underset{y_{\mathrm{t},1}\cdots y_{\mathrm{t},N}}{\arg\max} \sum_{i=1}^{N} \log(p(y_i|y_{\mathrm{t},i})). \tag{2.5}$$

This expression can be further simplified by using assumption 2 which states that the noise is additive. Due to this additive property $p(y_i|y_{\mathrm{t},i}) = p_{\mathrm{n}}(y_i - y_{\mathrm{t},i})$ in which the p_{n} indicates the noise probability density. This yields:

$$\hat{\mathbf{y}} = \underset{y_{\mathrm{t},1}\cdots y_{\mathrm{t},N}}{\arg\max} \sum_{i=1}^{N} \log(p_{\mathrm{n}}(y_i - y_{\mathrm{t},i})). \tag{2.6}$$

In this equation $y_{\mathrm{t},i}$ is a set of individual estimates of the observations that is varied. In the case of function approximation, we are not interested in a set of individual values that solve the maximisation problem, but we are interested in a function that approximates the data. Therefore, the set of points is replaced by a function with adjustable parameters:

$$\mathbf{b} = \underset{\mathbf{b}}{\arg\max} \sum_{i=1}^{N} \log(p_{\mathrm{n}}(y_i - \hat{y}(\mathbf{x}_i; \mathbf{b}))). \tag{2.7}$$

In this equation $\hat{y}(\mathbf{x}_i; \mathbf{b})$ is a function that approximates the individual values of $y_{\mathrm{t},i}$. $\mathbf{b}$ is a parameter vector that alters the approximation. This equation can be related to the empirical risk in (2.2). It follows that minimisation of the empirical risk with the cost function equated with the negative of the logarithm of the noise distribution, is identical to maximising the likelihood of the observations:

$$C(\hat{y}(\mathbf{x}_i; \mathbf{b}), y_i) = -\log(p_{\mathrm{n}}(y_i - \hat{y}(\mathbf{x}_i; \mathbf{b}))) \quad \Rightarrow$$
$$\text{minimising } R_{\mathrm{emp}} = \text{maximising the likelihood.} \tag{2.8}$$

Example 2.2 (Gaussian noise)
A well-known result is obtained if the additive noise is assumed to be zero-mean Gaussian noise. The probability density of Gaussian noise is given as

$$p_{\mathrm{n}}(\xi) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\xi^2}{2\sigma^2}\right). \tag{2.9}$$

Table 2.2: Common loss functions and their corresponding additive noise density, given in (Smola and Schölkopf, 1998)

	loss function	**density model**
ϵ-insensitive	$C(\xi) = \lvert\xi\rvert_\epsilon$	$p_\mathrm{n}(\xi) = \frac{1}{2+2\epsilon}\exp(-\lvert\xi\rvert_\epsilon)$
Laplacian	$C(\xi) = \lvert\xi\rvert$	$p_\mathrm{n}(\xi) = \frac{1}{2}\exp(-\lvert\xi\rvert)$
Gaussian	$C(\xi) = \frac{1}{2}\xi^2$	$p_\mathrm{n}(\xi) = \frac{1}{\sqrt{2\pi}}\exp\left(-\xi^2\right)$
Polynomial	$C(\xi) = \frac{1}{p}\lvert\xi\rvert^p$	$p_\mathrm{n}(\xi) = \frac{p}{2\Gamma(1/p)}\exp\left(-\lvert\xi\rvert^p\right)$

By equating the cost function with the noise distribution's negative logarithm, we obtain:

$$
\begin{aligned}
C(\xi) &= -\log(p_\mathrm{n}(\xi)) && \text{(2.10a)}\\
&= -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{\xi^2}{2\sigma^2}\right)\right) && \text{(2.10b)}\\
&= -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \log\left(\exp\left(-\frac{\xi^2}{2\sigma^2}\right)\right) && \text{(2.10c)}\\
&= -c + \left(\frac{\xi^2}{2\sigma^2}\right) && \text{(2.10d)}\\
&\propto \xi^2. && \text{(2.10e)}
\end{aligned}
$$

In this equation c is some constant and $\xi = y_i - \hat{y}(\mathbf{x}_i, \mathbf{b})$. So, (2.10) shows that a quadratic cost function results in an ML estimate for Gaussian additive noise. ∎

A table of common cost functions and their corresponding additive noise density is given in table 2.2. This table is copied from Smola and Schölkopf (1998). The ϵ-insensitive function used in the table, equals zero when ξ is smaller than ϵ, and increases linearly when ξ is larger than ϵ:

$$
\lvert\xi\rvert_\epsilon = \begin{cases} 0 & \text{if } \xi \leq \epsilon \\ \lvert\xi\rvert - \epsilon & \text{otherwise} \end{cases} \qquad (2.11)
$$

The ϵ-insensitive function is used in the Support Vector Machine (SVM), which will be treated in chapter 3.

2.1.4 Maximum A Posterior

The maximum likelihood estimate is closely related to the Maximum A Posterior estimation (MAP). The MAP estimate maximises the a posterior probability density:

$$\mathbf{b}_{\text{MAP}} = \arg\max_{\mathbf{b}} \prod_{i=1}^{N} p(\hat{y}(\mathbf{x}_i; \mathbf{b}) \,|y_i) \,. \tag{2.12}$$

The following relation is found by applying Bayes' theorem:

$$\arg\max_{\mathbf{b}} \prod_{i=1}^{N} p(\hat{y}(\mathbf{x}; \mathbf{b}) \,|y_i) = \arg\max_{\mathbf{b}} \prod_{i=1}^{N} \frac{p(y_i|\hat{y}(\mathbf{x}; \mathbf{b}))\, p(\hat{y}(\mathbf{x}; \mathbf{b}))}{p(y_i)}. \tag{2.13}$$

The $p(\hat{y}(\mathbf{x}; \mathbf{b}))$ denotes the a-priori probability density of the approximations. If all approximations are equally likely, e.g. no a-priori knowledge is present, (2.13) changes into:

$$\arg\max_{\mathbf{b}} \prod_{i=1}^{N} p(\hat{y}(\mathbf{x}; \mathbf{b}) \,|y_i) = \arg\max_{\mathbf{b}} \prod_{i=1}^{N} \frac{p(y_i|\hat{y}(\mathbf{x}_i; \mathbf{b}))}{p(y_i)}. \tag{2.14}$$

The maximisation of (2.14) is performed for $\mathbf{b}$, and because $p(y_i)$ is i.i.d., the optimal value of $\mathbf{b}$ is not influenced by $p(y_i)$, reducing (2.14) to (2.3). So, if all approximations are equally likely, the ML estimate is equal to the MAP estimate.

2.1.5 Regularisation

The minimisation of the empirical risk might give rise to a problem known as overfitting. Overfitting is the approximation of the realisation of the noise instead of the underlying function. The realisation of the noise is approximated if the amount of data is too small compared with the freedom of the approximator. The approximation predicts the output at the given samples (nearly) flawless while it typically fluctuates wildly in between these samples.

It is often known that these fluctuations of the approximation are incorrect. A second term that represents a priori knowledge on the function to approximate, can be included into the minimisation problem to impose e.g. smoothness. This second term is called a regularisation term. With this regularisation term, the new function to be minimised is given as:

$$R_{\text{emp}}(\mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} C(\hat{y}(\mathbf{x}_i; \mathbf{b}) \,, y_i) + \lambda D\hat{y}(\mathbf{x}; \mathbf{b}) \,. \tag{2.15}$$

In this equation D is some user-defined, problem-dependent operator on the approximation and λ a positive regularisation parameter. The value of λ indicates the confidence in the a-priori knowledge. If it is set to infinity, the samples do not have any influence, while setting it to zero ignores all a-priori knowledge.

The regularisation term is closely related to the Bayesian theory as shown in Poggio and Girosi (1989). Here, an informal presentation is given to show the idea. As before, Bayes' theorem states:

$$\prod_{i=1}^{N} p(y_i|\hat{y}(\mathbf{x}_i;\mathbf{b}))\, p(\hat{y}(\mathbf{x}_i;\mathbf{b})) = \prod_{i=1}^{N} p(\hat{y}(\mathbf{x}_i;\mathbf{b})\,|y_i)\, p(y_i). \tag{2.16}$$

The probability density of the approximation $\hat{y}(\mathbf{x}_i;\mathbf{b})$ is given as:

$$p(\hat{y}(\mathbf{x}_i;\mathbf{b})) = \varphi(\hat{y}(\mathbf{x}_i;\mathbf{b}))\,. \tag{2.17}$$

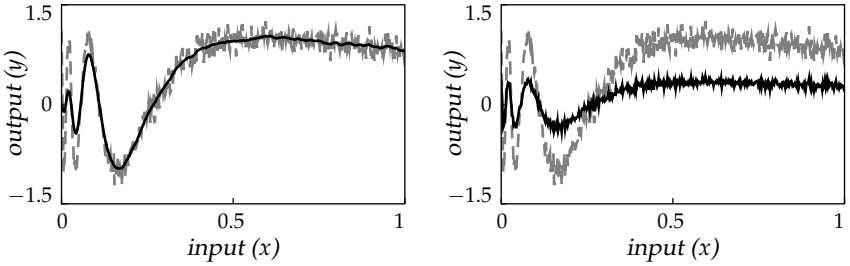
This density can be used to indicate that smooth functions are more likely to occur. Applying the logarithm to both sides yields:

$$\sum_{i=1}^{N} \log p(y_i|\hat{y}(\mathbf{x}_i;\mathbf{b})) + \sum_{i=1}^{N} \log \varphi(\hat{y}(\mathbf{x}_i;\mathbf{b})) = \sum_{i=1}^{N} \log p(\hat{y}(\mathbf{x}_i;\mathbf{b})\,|y_i) + \sum_{i=1}^{N} \log p(y_i) \tag{2.18}$$

The probability density of the approximation is equal for all the samples, simplifying the second left-hand term. Furthermore, the assumption of additive noise is included, which results in:

$$\sum_{i=1}^{N} \log p(y_i - \hat{y}(\mathbf{x}_i;\mathbf{b})) + N \log \varphi(\hat{y}(\mathbf{x};\mathbf{b})) - \sum_{i=1}^{N} \log p(y_i) = \sum_{i=1}^{N} \log p(\hat{y}(\mathbf{x}_i;\mathbf{b})\,|y_i) \tag{2.19}$$

Maximising the right-hand side of this equation is the same as the maximisation problem of the MAP estimate (2.12). The last term of the left-hand side can be omitted when maximising for $\mathbf{b}$, because it is a constant for all samples and independent of $\mathbf{b}$. The first term on the left-hand side equals the previous ML estimate (2.7). The second term on the left-hand side denotes the a-priori knowledge on the approximation and can be compared with the regularisation term of (2.15). Thus the MAP estimate



(a) Parameters influence derivative of the approximation

(b) Parameters influence absolute value of the approximation

Figure 2.3: Influence of Ridge regression

of the data with some prior knowledge on the likelihood of the approximation is equal with the ML estimate *plus* a regularisation term.

Example 2.3 (Ridge regression)
A regularisation method that is often used is *ridge regression* (Hoerl and Kennard, 1970). This method uses the two-norm of the parameter vector as regularisation term: $D\hat{y}(\mathbf{x};\mathbf{b}) = \mathbf{b}^{\mathrm{T}}\mathbf{b}$. The underlying a-priori knowledge if this norm is minimised, is that the parameters are likely to be zero. Due to the samples in the data set, this knowledge can be contradicted. However, the learning mechanism tries to keep the parameters small.

The influence of ridge regression is illustrated in figure 2.3. The gray line is the approximation without any form of regularisation. The sets of functions that can be approximated by the approximators in (a) and (b) are equal. However, how the set of function is formulated, is different; in (a) the parameters act on the derivative of the approximation, while in (b) they act on the absolute value of the approximation.

The consequence of minimising the parameters that act on the derivative in conjunction with the summed squared error, is that a smoother approximation is obtained. The corresponding disadvantage is that the fast fluctuations in the data for $x < 0.1$ cannot be approximated well. If the parameters act on the absolute value of the approximation, minimising the parameters will result in a too small an approximate. So, the influence of ridge regression depends on how the set of functions is constructed. ▩

2.2 Characterisation of function approximators

In the previous section the general approximation problem setting was stated as a minimisation problem. However, how we find an approximation was not mentioned. From the extensive literature in the field of function approximation, it can be deduced that there are a lot of possibilities. In order to characterise function approximators so as to deal with the great number of possibilities, several properties are examined. These properties are:

Minimisation criterion: What is the criterion the function approximator tries to minimise.

Set of functions: From what set of functions can the function approximator choose to minimise the above mentioned criterion.

Implementation: How does the function approximator find the correct function from the set of functions.

2.2.1 Minimisation criterion

The function approximator uses the minimisation criterion to determine whether one approximation is better than an other. A possible minimisation criterion is given in (2.15), which is the summed cost of the samples plus some regularisation term. The chosen cost function and the regularisation term, including the regularisation parameter λ, will have a significant influence on the final approximation that is obtained. In this thesis, we will be confronted with the following cost functions:

- quadratic cost function,
- ϵ-insensitive cost function.

The regularisation methods used are related to ridge regression. For more information of the influence of the cost function on the final approximation, see Huber (1981). Hoerl and Kennard (1970); Poggio and Girosi (1989, 1990); Tikhonov and Arsenin (1977) give more background about regularisation.

2.2.2 Set of functions

The learning mechanism selects a function out of a set of possible functions to approximate the data. Because the learning mechanism is limited to approximate the data by the functions out of this set, the set of possible functions co-determines the outcome of the approximation.

The set of functions is determined by the *allowed range of the parameters* and *the structure* on which these parameters act. The structure can be set beforehand and the learning mechanism has to tune the parameters to find the best approximation. Or, the structure itself can be adapted by the learning mechanism. In the second case, the learning mechanism has to select a structure and suitable values for the parameters to come to an approximation. A structure that is given beforehand and is unalterable during training, is called a *rigid structure*. A structure that can be altered throughout the training, is called a *flexible structure*.

Rigid structure

The function's structure is composed of elementary basis functions, in which the elementary functions are shaped by the parameters. In the case of a rigid structure, the composition of the elementary basis functions is set beforehand; so, only the shape of the basis functions can be altered by the learning mechanism.

Example 2.4
Assume we want to approximate some relation in the data by a fourth-order polynomial. The set of functions from which the learning mechanism has to select one is given as:

$$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4. \tag{2.20}$$

The elementary functions of this structure are the powers of x. These are summed to compose the structure. By altering the parameters b_j, the shape – in this case the amplitude – of the elementary functions is adapted. Throughout the training, only the parameters b_j are altered due to the samples (x_i, y_i). ◙

Flexible structure

When, next to the parameters, the structure is liable to adjustments, the learning mechanism has to locate the correct parameters *as well as* the correct structure.

Example 2.5 (Limited Gaussian functions)
Let us assume that we would like to approximate the data by a weighted sum of Gaussian functions. The Gaussian function is given as:

$$f_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\sigma^2}\right). \tag{2.21}$$

As to the example, the width σ of this function is fixed. The centres, c_i, are selected to coincide with the inputs of the training data. Because the memory is limited, we wish to select only 10 centres.

The individual Gaussian functions with their centre on some training sample are the elementary functions of which the structure of possible functions is composed. By means of trial and error, the learning mechanism selects ten of these centres, i.e. it selects an appropriate structure. The parameters that have to be given a value are the weights of the weighted sum.

Instead of searching for apt Gaussian functions on the training samples and thus adapting the structure, the same set of functions can be approximated by using ten Gaussian functions and use the centres as parameters that can be tuned. The adaption of the structure is replaced by introducing extra parameters, thus obtaining a rigid structure. ▩

Because the same set of functions can be obtained by using a rigid structure with some extra parameters as in a flexible structure, as shown in example 2.5, the structure and the parameters cannot be defined in an unequivocal way by the set of functions they span, and are merely determined on *how* the problem is looked at. However, they can be defined by how they act in the learning mechanism: adaptations of the structure can only be done by including or removing terms, while parameters can alter in a continuous fashion. This is summarised in the following two definitions:

Definition 1 (Structure) *The structure of a function is the composition of elementary basis functions; adapting the structure can only be done by adding or removing terms.*

Definition 2 (Parameter) *A parameter is a variable that shapes the elementary basis function of which the structure is composed; furthermore, a parameter can be adapted by the learning mechanism.*

2.2.3 Implementation

The optimal approximation is completely determined by the minimisation criterion and the set on which the minimisation is carried out. How this approximation is found in the allowed set of functions *should* not be of any consequence to the outcome; it merely indicates the search process.

Nonetheless, this search process is an important property of the learning mechanism, because locating the best function is not straightforward, especially if the optimisation problem is non-convex (Boyd and Vandenberghe, 2004). For this reason, an acceptable function found quickly, can be more satisfactory than the best function found after a long search.

We can look at the update algorithm contained in the learning mechanism to characterise the way in which the learning mechanism finds

an approximation. We call those calculations an update step that are required for a new estimate. If several iterations are required to obtain a new update, e.g. due to some internal optimisation routine, this will still be denoted as just one update step, because the intermediate results are not considered estimates. We will differentiate between two classes of updates: single step and recursive.

The first class uses all the data to come to an approximation in one step. This method is used, e.g., by Ordinary Least Squares (OLS), which calculates the parameter vector explicitly in one step by the use of the normal equations, see, among others, (Björck, 1996). The Support Vector Machine (SVM) can also be classified as an approximator that uses only one update step. The SVM will be treated in the next chapter. To obtain an approximation in the SVM setting, a convex optimisation problem needs to be solved. This requires some iterations, but in between, no new estimate is presented. These iterations are therefore not considered update steps.

The second class consists of learning mechanisms that update their approximation in successive steps, and so obtain an approximation in a recursive fashion. A rather broad set of methods use such an update scheme. It contains methods that incorporate explicitly *all* the information of the previously presented samples, as well as methods that incorporate *none* of the samples previously supplied explicitly for an update. To incorporate information of previous samples for updating, some kind of storage is required. This storage has to be updated in conjunction with the approximation. A function approximator that uses all the information of previously presented samples is the method of Recursive Least Squares (RLS).

Whenever a new sample becomes available, it is incorporated into the present approximation, resulting in an updated approximation. To incorporate the sample in the new approximation without forgetting the previous samples, the previous samples are stored in a correlation matrix. This matrix is updated with every new sample.

A method that only uses the present sample without the explicit use of previous samples is an Multilayer Perceptron neural network (MLP) which uses the backpropagation rule to update its parameters (Haykin, 1994; Zurada, 1992). The gap between storing the information of all the previous samples and storing no information at all, is filled by methods that use *some* information of the previous samples. One can think of the use of the momentum in the field of Artificial Neural Networks (ANN) (Haykin, 1994; Zurada, 1992).

Table 2.3: Properties of the recursive least squares method and the support vector machine

	Minimisation criterion
RLS	$\sum_{i=1}^{N}(\hat{y}(\mathbf{x}_i;\mathbf{b})-y_i)^2+\lambda\mathbf{b}^{\mathrm{T}}\mathbf{b}$
SVM	$\sum_{i=1}^{N}\lvert\hat{y}(\mathbf{x}_i;\mathbf{b})-y_i\rvert_\epsilon+\lambda\mathbf{b}^{\mathrm{T}}\mathbf{b}$
	Set of structures
RLS	Rigid structure that has to be decided on beforehand
SVM	Flexible structure that depends on the data instead of a predefined structure
	Update step
RLS	$\mathbf{K}:=\mathbf{K}-\frac{\mathbf{K}\mathbf{x}^{\mathrm{T}}\mathbf{x}\mathbf{K}}{1+\mathbf{x}\mathbf{K}\mathbf{x}^{\mathrm{T}}}$ $\mathbf{b}:=\mathbf{b}+\mathbf{K}\mathbf{x}^{\mathrm{T}}(y-\mathbf{x}\mathbf{b})$
SVM	By use of convex optimisation

Example 2.6 (Comparison of two function approximators)
The properties treated above give us the tools to deal with the differences of the vast variety of function approximators. This can be illustrated by comparing RLS and SVM. Both methods will be treated in more detail hereafter, but by investigating the criterion, the set of functions and the update step, some idea can be obtained of these methods. These characteristics are summarised in table 2.3. Based on these, several observations can be made:

- Due to the different cost functions both methods will solve a different problem. The quadratic cost function will give an ML estimation for Gaussian noise, which makes it sensitive to outliers. The ϵ-insensitive cost function is less sensitive to outliers. However, it allows for small errors.
- Due to the rigid structure of the RLS method, a-priori knowledge is required to form this structure. If no a-priori knowledge is available, a general structure can be used.
- The SVM method uses a flexible structure. This allows for adaptation on both parameters and structure, resulting in a broad set of functions that can be approximated. The structure has to be determined together with the parameters, which will result in a higher computational load.
- The RLS method updates its prediction at every step while remembering previous information. This makes it applicable for situations in which the

data is received successively and an estimate is required at every step, like in control or adaptive filters. The previous information is stored in the matrix **K** and contains all the information necessary to update the solution in such a way that the solution found is equal to the solution obtained when all the samples were used.

- The calculations for the SVM have to be done in a batch. This makes it applicable if all the data is present beforehand.

■

2.3 Conditions

The function approximator is embedded in a learning control setting. When placed in a control setting, the prediction of the approximator is injected into the controlled system as a signal. In the LFFC scheme this signal acts as a feedforward control signal and should steer the plant with the desired response. Because the approximator is included in the system and it injects signals into it, conditions are imposed on its behaviour. The conditions for the approximator's behaviour is the subject of this section.

In Velthuis (2000) several requirements were formulated with respect to function approximator. Notwithstanding that these conditions were formulated for one individual scheme (LFFC), they do suggest that several conditions should hold for a function approximator to be applicable in any control scheme. Generalising from the above-mentioned source, two main conditions can be formulated in which further subconditions can be subsumed:

1) *Real-time constraints:* the approximator should be implementable in an (embedded) computer operating real-time.

2) *Generalisation ability:* the approximator should give a meaningful approximation for any relevant input, based on the samples supplied before.

The first of these conditions originates from the limited resources available on a computer. Nowadays nearly all controllers are implemented digitally and therefore the limitations of the available hardware should be taken into account. This limits the computation time as well as the amount of available memory the approximator may use. For the condition on limited resources to hold, several subconditions have to hold. These can be formulated as follows:

1a) The computation time of a prediction has to be smaller than the sampling period; for the applications we are interested in, this is approximately 1 [ms].

1b) The memory requirements of the approximator have to be limited.

The first of these conditions states that the computation time of the approximator should be less than a sample period. This is a necessary condition, because in one sample period, additional calculations have to be made.

If the update is done on-line, then the time allowed for the calculation of the update depends on the implementation of the control setting: the first option is that the update is calculated in the real-time loop. The calculations for the update have to be made within a sample period, *together with the other calculations in the real-time loop.* The second option is to put the calculation of the update outside the real-time loop. This allows for updates that need more calculation time than available in a sample period. As long as the function approximator is processing a sample, samples supplied newly will *not* be processed and are omitted. The calculations are executed when the real-time loop does not require processor time, and care should therefore be taken, that the function approximator gets enough calculation time from the processor to obtain an adequate approximation for the learning controller.

Condition 2 states that the approximator should give meaningful predictions based on the samples. The following sub-conditions stem from this condition:

2a) The approximator should generalise well.

2b) The approximator should be able to cope with noisy training data.

2c) The approximator should be able to approximate functions with the appropriate number of inputs.

Generalisation in subcondition 2a indicates that the outputs of newly encountered samples are predicted well, and furthermore, that samples that are more likely to occur should be predicted better. The same was striven for by the minimisation of the risk function, repeated below:

$$R(\mathbf{b}) = \int_{-\infty}^{\infty} C(\hat{y}(\mathbf{x};\mathbf{b}), y)\ \mathrm{d}P(\mathbf{x}, \mathbf{x}_\mathrm{u}, y) \tag{2.1}$$

Based on this minimisation criterion, generalisation is a rather straightforward term, indicating that samples taken with the probability distribution $P(\mathbf{x}, \mathbf{x}_\mathrm{u}, y)$ will give a small cost on the average (White, 1989).

However, one can only give a good prediction for an arbitrary input if some a-priori knowledge on the relation is included. If no knowledge is assumed on the relation, a prediction cannot be obtained for a new input: all possible predictions are equally unlikely. An assumption that is commonly made, is the assumption that the relation is smooth (Haykin, 1994; Poggio and Girosi, 1990). This assumption is also used throughout this thesis.

Assumption 3 (Smoothness) *The relation that is sought is smooth.*

Condition 2b is of importance, because noise is inherently present. Even with this noise the learning mechanism has to be able to find correct parameters, *as well as a correct structure*.

Conditions 2c is included because the number of inputs can become considerable. In previous work on LFFC the function approximator could not manage this and this hindered the extension to higher-dimensional input spaces (De Vries et al., 2001; Otten et al., 1997; Starrenburg et al., 1996; Velthuis, 2000; Velthuis et al., 1998). In the introduction this was shortly mentioned as the curse of dimensionality. The curse indicates that the number of parameters needed for approximation grows exponentially with the input dimension. This results in large memory requirements, difficult training and a bad generalisation (Brown and Harris, 1994; De Vries et al., 2001). Learning mechanisms that divide the input space into small regions are prone to the curse of dimensionality.

2.3.1 Desired properties

Next to the conditions given above, there are several desirable properties that make the approximator easier to use in practice. The first desired property is that the method should be capable of incorporating a-priori knowledge. This can be done in several ways. Two possibilities are regularisation or limiting the allowed set of functions. Other methods can be thought of, see e.g. the paper of Schölkopf, Simard, Smola, and Vapnik (1998) in which the kernel is adapted to represent the a-priori knowledge.

A second desired property is that the contents of the function approximator provides useful information. This makes it possible to check whether the approximator has approximated what was intended, and to see if its predictions are not unreliably large, with the possibility of damaging the plant. A clear example of the advantage when the contents of the network gives useful information, is in the case of parsimonious learning feedforward control (De Vries et al., 2001).

The last desired property is that the method is easy to use. The tuning of the parameters for the approximator should be intuitive and clear. The tuning of the parameters should not become an optimisation problem in itself.

Three

Off-line function approximation

FUNCTION APPROXIMATORS ARE treated in this and in the next chapter. This chapter treats off-line function approximation, while the next treats on-line approximation. Off-line approximation refers to those learning mechanisms that require all the data to be available before making an approximation.

This chapter starts by treating several off-line function approximators. The advantages and disadvantages of these function approximators are discussed in section 3.2. Based on this discussion, a proposition for a new function approximator is made in section 3.3. First, the new method is evaluated, and second, the method is compared with other learning mechanisms. The evaluation and comparison are done in section 3.4 and 3.5 respectively. At the end of this chapter, a review of the chapter is given, including a discussion about whether this method can be used in learning control or not.

3.1 Off-line methods

Due to the vast amount of literature on function approximation, a selection is made as to which methods will be treated. Criteria used for the selection, as well as the methods resulting from these are:

Previously used in LFFC research: B-Spline Networks,

Commonly used in learning control: Ordinary Least Squares, Artificial Neural Networks, Radial Base Function Networks,

Promising to avoid the curse of dimensionality: Support Vector Machines, Least Squares Support Vector Machines,

Explanatory reasons: Dual Least Squares.

Methods whose theory is used are treated in more depth than those which are only used for comparison.

3.1.1 Ordinary Least Squares

The Ordinary Least Squares (OLS) method tries to find a linear relation between a set of fixed indicator functions $f_i(\mathbf{x}), i = 1 \ldots n$ and the output $\hat{y}$ of the form:

$$\hat{y}_{\text{ls}}(\mathbf{x}) = b_1 f_1(\mathbf{x}) + b_2 f_2(\mathbf{x}) + \ldots + b_n f_n(\mathbf{x}). \tag{3.1}$$

By selecting a set of n functions and an allowable range for $\mathbf{b}$, the set of functions that can be approximated is determined. The functions $f_i(\mathbf{x})$ are called *indicators* or *basis functions* and can be non-linear functions of the input vectors. These functions implement a mapping from the input space to a *feature space*. A linear approximation is made in this feature space.

The value of the parameter vector $\mathbf{b}$, containing the elements b_i, follows from the samples given. The use of the quadratic cost function coincides with a maximum likelihood (ML) estimate for additive Gaussian noise as seen in section 2.1.3.

Define matrix $\mathbf{X}$, column-vectors $\mathbf{y}$ and $\mathbf{b}$ containing respectively the indicators for all the N samples, the target values and the parameters:

$$\mathbf{X} = \begin{bmatrix} f_1(\mathbf{x}_1) & f_2(\mathbf{x}_1) & \cdots & f_n(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_N) & f_2(\mathbf{x}_N) & \cdots & f_n(\mathbf{x}_N) \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{3.2}$$

in which the subscript of $\mathbf{x}$ and y indicates the sample number. The target vector $\mathbf{y}$ is assumed to be corrupted by i.i.d. noise, denoted by $\boldsymbol{\varepsilon}$. Using the matrix notation introduced above, the following relation holds:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \boldsymbol{\varepsilon} \tag{3.3}$$

with

$$E(\boldsymbol{\varepsilon}) = 0 \quad \text{and} \quad E(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^{\mathrm{T}}) = \sigma^2 \mathbf{I}. \tag{3.4}$$

The minimisation of the summed squared approximation error is given as:

$$\min_{\mathbf{b}} \left(\tfrac{1}{2} \left\| \mathbf{X}\mathbf{b} - \mathbf{y} \right\|_2^2 + \tfrac{1}{2}\lambda \left\| \mathbf{b} \right\|_2^2 \right). \tag{3.5}$$

The term $\frac{1}{2} \left\| \mathbf{b} \right\|_2^2$ is the regularisation term that minimises the parameters and λ is a positively valued regularisation parameter.

The solution of this minimisation problem can be found by equating the derivatives with respect to $\mathbf{b}$ with zero, resulting in the well-known normal equations, see e.g. (Björck, 1996; Draper and Smith, 1998; Stewart, 1998):

$$\begin{aligned} \tfrac{1}{2} \frac{\mathrm{d}\left(\left\| \mathbf{X}\mathbf{b} - \mathbf{y} \right\|_2^2 + \lambda \left\| \mathbf{b} \right\|_2^2 \right)}{\mathrm{d}\mathbf{b}} &= \tfrac{1}{2} \frac{\mathrm{d}\left((\mathbf{X}\mathbf{b} - \mathbf{y})^{\mathrm{T}} (\mathbf{X}\mathbf{b} - \mathbf{y}) + \lambda \mathbf{b}^{\mathrm{T}}\mathbf{b} \right)}{\mathrm{d}\mathbf{b}} \\ &= \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{b} + \lambda \mathbf{b} - \mathbf{X}^{\mathrm{T}}\mathbf{y}. \end{aligned} \tag{3.6}$$

Equating this derivative with zero yields:

$$\left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{I} \right) \mathbf{b} = \mathbf{X}^{\mathrm{T}}\mathbf{y}, \tag{3.7}$$

in which $\mathbf{I}$ is the identity matrix. To solve the normal equations, it is necessary to solve a system of equations with the size of the number of indicators, n. The size of the system is independent of the number of samples supplied to it, N, which makes it attractive in situations when there are few indicators but many samples.

When the parameter vector $\mathbf{b}$ is known, the output for a new sample $\mathbf{x}_{\mathrm{new}}$ can be predicted. Based on (3.1), the prediction is calculated as:

$$\hat{y}_{\mathrm{ls}}(\mathbf{x}_{\mathrm{new}}) = \sum_{i=1}^{n} b_i f_i(\mathbf{x}_{\mathrm{new}}) \tag{3.8}$$

or, if the matrix presentation is preferred,

$$\hat{y}_{\mathrm{ls}}(\mathbf{x}_{\mathrm{new}}) = \mathbf{f}(\mathbf{x}_{\mathrm{new}})\mathbf{b} \tag{3.9}$$

in which $\mathbf{f}$ is a *row*-vector, with the values of the indicator functions for $\mathbf{x}$: $\mathbf{f}(\mathbf{x}_{\mathrm{new}}) = [f_1(\mathbf{x}_{\mathrm{new}}), f_2(\mathbf{x}_{\mathrm{new}}), \ldots, f_n(\mathbf{x}_{\mathrm{new}})]$.

Example 3.1
Let us illustrate the use of ordinary least squares with an example. The data for which a relation is sought, is data set 1. For this example, we collect 25 noise-free samples whose inputs are evenly distributed on the input space. We want to

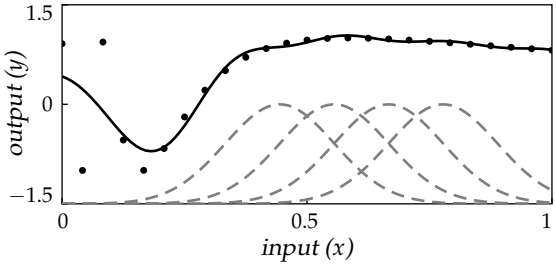


Figure 3.1: Approximation with a radial base function with fixed width and centres

approximate these samples with 10 indicator functions. The indicator functions are chosen to be Gaussian functions with equidistant centres:

$$f_i(x) = \exp\left(-\frac{(x-c_i)^2}{2\sigma^2}\right) \tag{3.10}$$

with

$$c_i = \frac{i}{n-1} - \frac{1}{n-1}, \quad i = 1, \ldots, 10. \tag{3.11}$$

σ is equated with the distance between the centres, $\sigma = 0.11$. The matrices $\mathbf{X}$ and $\mathbf{y}$ are formed, based on this data. The first elements of these are given as:

$$\mathbf{X} = \begin{bmatrix} 1.00 & 0.61 & 0.14 & 0.01 & 0.00 & \ldots \\ 0.93 & 0.82 & 0.27 & 0.03 & 0.00 & \ldots \\ 0.75 & 0.97 & 0.46 & 0.08 & 0.01 & \ldots \\ 0.53 & 0.99 & 0.68 & 0.17 & 0.02 & \ldots \\ 0.32 & 0.88 & 0.88 & 0.32 & 0.04 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 0.91 \\ -1.00 \\ 0.94 \\ -0.54 \\ -1.00 \\ \vdots \end{bmatrix}. \tag{3.12}$$

The size of matrix $\mathbf{X}$ is 25×10 and $\mathbf{y}$ is 25×1. The normal equations (3.7) are used to calculate the parameter vector $\mathbf{b}$, based on these matrices. With the parameter vector a prediction can be made (3.8), see figure 3.1. In this figure, the black dots are the training samples, the black line the approximation and the four gray Gaussian functions are four out of the ten indicator functions. The approximation with these Gaussian functions is not good at small values of x. ▩

3.1.2 Dual Least Squares

Instead of equating the derivatives of the minimisation criterion with zero as in (3.7), the criterion can also be minimised by the use of optimisation theory, e.g. (Aoki, 1971). This results in a dual representation

of the least squares problem. The minimisation problem of (3.5) can be rewritten to a standard form for optimisation:

$$\min_{\mathbf{b}} \left(\tfrac{1}{2}\mathbf{e}^{\mathrm{T}}\mathbf{e} + \tfrac{1}{2}\lambda \mathbf{b}^{\mathrm{T}}\mathbf{b} \right) \tag{3.13}$$

with equality constraint

$$\mathbf{y} - \mathbf{X}\mathbf{b} - \mathbf{e} = 0, \tag{3.14}$$

in which $\mathbf{e}$ is a column-vector containing the approximation error for all the samples. This form allows a Lagrangian to be constructed by which the minimisation problem can be solved, see appendix A.1. The Lagrangian is given as:

$$\mathcal{L} = \tfrac{1}{2}\mathbf{e}^{\mathrm{T}}\mathbf{e} + \tfrac{1}{2}\lambda \mathbf{b}^{\mathrm{T}}\mathbf{b} + \boldsymbol{\alpha}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{b} - \mathbf{e}). \tag{3.15}$$

The vector $\boldsymbol{\alpha}$ consists of the Lagrangian multipliers and has the same dimension, N, as the number of samples. The solution of the minimisation problem can be found by equating the derivatives of the Lagrangian with respect to $\mathbf{b}, \mathbf{e}$ and $\boldsymbol{\alpha}$ with zero:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \lambda \mathbf{b} - \mathbf{X}^{\mathrm{T}}\boldsymbol{\alpha} = 0, \tag{3.16a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{e}} = \mathbf{e} - \boldsymbol{\alpha} = 0, \tag{3.16b}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}} = \mathbf{y} - \mathbf{X}\mathbf{b} - \mathbf{e} = 0. \tag{3.16c}$$

The solution of the optimisation problem can be formulated in the primal variable $\mathbf{b}$, or it can be specified in the dual variable $\boldsymbol{\alpha}$. This is achieved by elimination through substitution of $\mathbf{e}, \boldsymbol{\alpha}$ or of $\mathbf{e}, \mathbf{b}$ respectively in (3.16). In the primal form it is found that:

$$\left(\lambda \mathbf{I} + \mathbf{X}^{\mathrm{T}}\mathbf{X} \right) \mathbf{b} = \mathbf{X}^{\mathrm{T}}\mathbf{y}, \tag{3.17}$$

which is, as expected, identical to the solution found in (3.7). In the dual form, the solution is given as:

$$\left(\mathbf{I} + \frac{1}{\lambda}\mathbf{X}\mathbf{X}^{\mathrm{T}} \right) \boldsymbol{\alpha} = \mathbf{y}, \tag{3.18a}$$

$$\mathbf{b} = \frac{\mathbf{X}^{\mathrm{T}}\boldsymbol{\alpha}}{\lambda}. \tag{3.18b}$$

This can be slightly rewritten for convenience, by using $\boldsymbol{\alpha} := \boldsymbol{\alpha}/\lambda$:

$$\left(\lambda \mathbf{I} + \mathbf{X}\mathbf{X}^{\mathrm{T}}\right) \boldsymbol{\alpha} = \mathbf{y}, \tag{3.19a}$$

$$\mathbf{b} = \mathbf{X}^{\mathrm{T}} \boldsymbol{\alpha}. \tag{3.19b}$$

The matrix product $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ contains the innerproducts of all the training samples with the other training samples in the feature space:

$$\left[\mathbf{X}\mathbf{X}^{\mathrm{T}}\right]_{ij} = \mathbf{f}(\mathbf{x}_i)\mathbf{f}(\mathbf{x}_j)^{\mathrm{T}}. \tag{3.20}$$

The number of equations corresponds to the number of samples. This time, it is independent of the number of indicators.

The prediction for the dual least squares for a new input $\mathbf{x}_{\mathrm{new}}$ can be given in terms of matrices:

$$\hat{y}_{\mathrm{d}}(\mathbf{x}_{\mathrm{new}}) = \mathbf{f}(\mathbf{x}_{\mathrm{new}})\, \mathbf{b} \tag{3.21a}$$

$$= \mathbf{f}(\mathbf{x}_{\mathrm{new}})\, \mathbf{X}^{\mathrm{T}} \boldsymbol{\alpha}, \tag{3.21b}$$

or in the form of a summation:

$$\hat{y}_{\mathrm{d}}(\mathbf{x}_{\mathrm{new}}) = \sum_{i=1}^{n} b_i f_i(\mathbf{x}_{\mathrm{new}}) \tag{3.22a}$$

$$= \sum_{i=1}^{N} \langle \mathbf{f}(\mathbf{x}_{\mathrm{new}}), \mathbf{f}(\mathbf{x}_i) \rangle \alpha_i. \tag{3.22b}$$

Note that the prediction with the use of $\boldsymbol{\alpha}$ (3.22b), as well as the calculation of $\boldsymbol{\alpha}$ (3.19a), *do not* depend on the individual elements of the feature space, but solely on the innerproducts in the feature space. Therefore, large or even infinite dimensional feature spaces can be used. The large dimensional feature space can be illustrated with an approximation by polynomials. The infinite dimensional feature space is illustrated later on in example 3.3

Example 3.2 (Large feature spaces)
Let us assume that we wish to approximate a relation based on two inputs with a second order polynomial. The sought relation is of the form:

$$\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1 x_2 + b_4 x_1^2 + b_5 x_2^2. \tag{3.23}$$

The indicator function in this approximation is given as:

$$\mathbf{f}(\mathbf{x}) = \left[1, x_1, x_2, x_1 x_2, x_1^2, x_2^2\right]. \tag{3.24}$$

The innerproduct in the feature space between two vectors becomes:

$$\mathbf{f}(\mathbf{x})\mathbf{f}^{\mathrm{T}}(\mathbf{y}) = 1 + y_1x_1 + y_2x_2 + y_1y_2x_1x_2 + y_1^2x_1^2 + y_2^2x_2^2 = (\langle \mathbf{x}, \mathbf{y} \rangle + 1)^2 . \quad (3.25)$$

When the degree of the approximation or the number of inputs increases, the number of indicator elements swiftly increases in (3.24). As indicated by the last term of (3.25), the innerproduct of the indicator functions can still be easily calculated. ▩

3.1.3 Support Vector Machines

The support vector machines were introduced by Vapnik (2000) for classification and regression. This method is based on statistical learning theory and for more on the subject, see (Campbell, 2000, 2002; Cristianini and Shawe-Taylor, 2000; Schölkopf, Mika, Burges, Knirsch, Müller, Rätsch, and Smola, 1999; Smola, 1998; Vapnik, 2000).

The set of functions is again described by the representation of (3.1), although the offset is explicitly denoted. This yields the following structure:

$$\hat{y}_{\mathrm{svm}} = \mathbf{f}(\mathbf{x})\mathbf{b} + b_0. \quad (3.26)$$

The cost function that is used, is not the quadratic cost function, but the ϵ-insensitivity function as given in (2.11). This allows for errors smaller than ϵ to go unpunished. This cost function has the following properties relative to the quadratic cost function:

- Less sensitive to outliers,
- Results in a sparse solution.

The diminished sensitivity of the ϵ-insensitive cost function is given in (Huber, 1981) and can be explained by the noise distribution for which this cost function will give an ML estimate. The second property will be made clear in this section.

The minimisation problem is formulated as:

$$\min_{\mathbf{b},b_0} \sum_{i=1}^{N} |\mathbf{f}(\mathbf{x})\mathbf{b} + b_0 - y_i|_\epsilon + \frac{\lambda}{2}\mathbf{b}^{\mathrm{T}}\mathbf{b}. \quad (3.27)$$

This minimisation problem is rewritten into a form that can be solved, using optimisation theory. The equivalent problem is defined as:

$$\min_{\mathbf{b},b_0} C \sum_{i=1}^{N} (\xi_i + \xi_i^*) + \tfrac{1}{2}\mathbf{b}^{\mathrm{T}}\mathbf{b}, \quad (3.28)$$

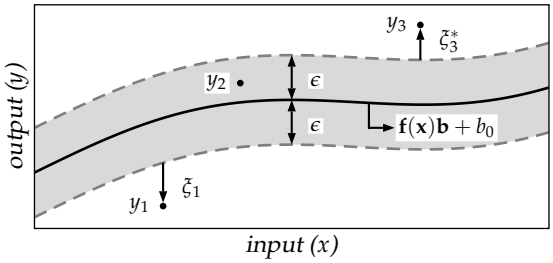


Figure 3.2: Variables used in equation 3.29

with the following constraints:

$$\begin{aligned}
\mathbf{f}(\mathbf{x})\mathbf{b} + b_0 - y_i &\leq \epsilon + \xi_i, && \text{(3.29a)}\\
y_i - \mathbf{f}(\mathbf{x})\mathbf{b} - b_0 &\leq \epsilon + \xi_i^*, && \text{(3.29b)}\\
-\xi_i^* &\leq 0, && \text{(3.29c)}\\
-\xi_i &\leq 0. && \text{(3.29d)}
\end{aligned}$$

The variables used in these equations are illustrated in figure 3.2. The $\xi_i^{(*)}$ are slack variables and are used to construct the ϵ-insensitive band. ξ^* represents the distance from a training sample to the upper boundary of this band. If the value of the target is smaller than the value for the prediction *plus* the allowable error ϵ, then this slack variable will be zero. ξ works just the other way round. So, in the figure ξ_1 will have a non-zero value because the target is smaller than the prediction minus the insensitivity zone. The value of ξ_1^* will be zero because the value of the training sample is smaller than the upper-side of the band. For the training sample y_2 both slack variables are zero and finally, for y_3 only ξ_3^* will be non-zero.

In the notation of the minimisation problem, C takes the place of λ, to be compatible with the SVM literature. C and λ are inversely proportional to each other. This optimisation problem can be solved by using the Kuhn-Tucker theorem, which is stated together with the calculations, in appendix A.
The result is a dual optimisation problem:

$$\max_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \sum_{i=1}^{N} (\alpha_i^* - \alpha_i)\, y_i - \epsilon \sum_{i=1}^{N} (\alpha_i^* + \alpha_i) - \frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i^* - \alpha_i)\,(\alpha_j^* - \alpha_j) \mathbf{f}(\mathbf{x}_i)\mathbf{f}^{\mathrm{T}}(\mathbf{x}_j). \quad \text{(3.30)}$$

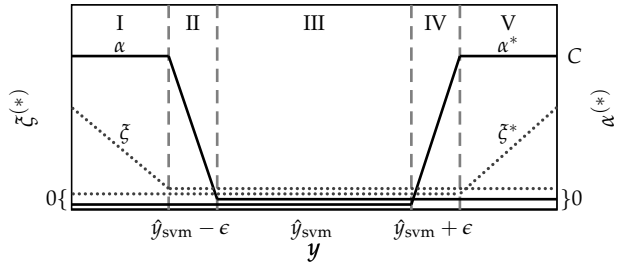


Figure 3.3: Relation between Lagrangian multipliers and slack variables

In this equation $\alpha^{(*)}$ are Lagrangian multipliers. The following constraints should be fulfilled so that the solution is feasible:

$$0 \leq \alpha_i \leq C, \tag{3.31a}$$

$$0 \leq \alpha_i^* \leq C, \tag{3.31b}$$

$$\sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0. \tag{3.31c}$$

The Karush-Kuhn-Tucker conditions (KKT) given below, hold at the optimum (Cristianini and Shawe-Taylor, 2000):

$$\alpha_i(\mathbf{f}(\mathbf{x}_i)\mathbf{b} + b_0 - y_i - \epsilon - \xi_i) = 0, \tag{3.32a}$$

$$\alpha_i^*(y_i - \mathbf{f}(\mathbf{x}_i)\mathbf{b} - b_0 - \epsilon - \xi_i^*) = 0, \tag{3.32b}$$

$$\xi_i^* \xi_i = 0, \tag{3.32c}$$

$$\alpha_i^* \alpha_i = 0, \tag{3.32d}$$

$$(\alpha_i - C)\xi_i = 0, \tag{3.32e}$$

$$(\alpha_i^* - C)\xi_i^* = 0. \tag{3.32f}$$

This convex optimisation problem can be solved efficiently by the SMO algorithm (Mangasarian and Musicant, 1998; Shevade, Keerthi, Bhattacharyya, and Murthy, 1999).

The relation between the slack variables and the Lagrangian multipliers is illustrated in figure 3.3. This figure is a cross-section of figure 3.2 for some value of x. In region I the training sample is smaller than the prediction minus the insensitivity zone. As noted before, the value of the slack variable ξ has a positive value in this situation. Due to the KKT condition (3.32e), it follows that the corresponding Lagrangian multiplier α is equal to C.

In region II the training sample is equal to the prediction minus the insensitivity zone. Actually, this is not a region but a line. Due to the

same KKT condition, the value of α can take a value different from C. Due to the conditions for a feasible solution (3.31a) the value should be between zero and C.

In the remaining regions ξ is zero. Therefore, it follows from (3.32a) that α is zero. The slack variables and Lagrangian multipliers α^* and ξ^* can be found by similar reasoning.

In region II the value of α is non-zero and ξ is zero. Incorporating this in (3.32a) and (3.32b), b_0 can be calculated:

$$b_0 = y_i - \mathbf{f}(\mathbf{x}_i)\mathbf{b} - \epsilon \quad \text{for} \quad 0 < \alpha_i < C, \tag{3.33a}$$

$$b_0 = y_i - \mathbf{f}(\mathbf{x}_i)\mathbf{b} + \epsilon \quad \text{for} \quad 0 < \alpha_i^* < C. \tag{3.33b}$$

When the bias and the parameter vector are known, we can calculate the prediction:

$$\hat{y}_{\text{svm}}(\mathbf{x}_{\text{new}}) = \mathbf{f}(\mathbf{x}_{\text{new}})\mathbf{b} + b_0 \tag{3.34a}$$

$$= \mathbf{f}(\mathbf{x}_{\text{new}}) \sum_{i=1}^{N} (\alpha_i^* - \alpha_i)\, \mathbf{f}^{\text{T}}(\mathbf{x}_i) + b_0 \tag{3.34b}$$

$$= \sum_{i=1}^{N} (\alpha_i^* - \alpha_i)\, \mathbf{f}(\mathbf{x}_{\text{new}})\mathbf{f}^{\text{T}}(\mathbf{x}_i) + b_0 \tag{3.34c}$$

$$= \sum_{i=1}^{N} (\alpha_i^* - \alpha_i)\, \langle \mathbf{f}(\mathbf{x}_{\text{new}}), \mathbf{f}(\mathbf{x}_i) \rangle + b_0. \tag{3.34d}$$

In the transition from (3.34a) to (3.34b) the equality (A.15) of appendix A is used. The same observation as was made for the dual least squares can be made for SVM: for the calculations of the Lagrangian multipliers as well as for the prediction of the output, the indicator vectors are only used in innerproducts and the individual elements in the feature space are never used. This makes it possible to use a large, or even an infinite, dimensional feature space. The advantage of using the innerproduct instead of the individual elements becomes significant when the number of elements becomes large and the calculation of the innerproduct is fast.

In the field of SVM, the innerproduct of two samples in the feature space is called a kernel function: $k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{y}) \rangle$. Numerous kernel functions are known that can use splines, polynomials, radial base functions, sigmoidal functions or a user specific function to find a relation in the data. Mercer's theorem gives conditions how to test whether a kernel is a valid innerproduct in some space (Mercer, 1909). This can be used to test whether a kernel function is a valid innerproduct when it is not explicitly created from an innerproduct.

The prediction for a new sample is a weighted sum of the innerproducts with the training samples as given in (3.34d). The weights are given by the Lagrangian multipliers, and when these are zero, the innerproduct with the corresponding training sample does not have any influence on the prediction. This makes the training sample superfluous for prediction and it can safely be forgotten after the training phase. The weight in the summation is zero if both α_i^* and α_i are zero, which happens in the insensitivity zone. The training vectors that have a non-zero weight are called *support vectors*. This is a subset of the original training set, and only this limited number of training samples has to be stored for the approximation.

Because the innerproduct is used in the feature space and not the individual elements, the dimension of the feature space does not matter.

> *This means that the SVM can use an infinite dimensional feature space, and the data is summarised by means of only a limited set of training samples and their corresponding Lagrangian multipliers!*

Example 3.3 (Infinite splines)
To illustrate the use of this statement, we will approximate the samples of data set 1 with a piecewise linear function. A piecewise linear function can be described by first order splines. The knots of these splines are at $x_{\mathrm{k},1}, \ldots, x_{\mathrm{k},n}$. The indicator function for this approximation is written as:

$$\mathbf{f}(x) = \left[1, (x - x_{\mathrm{k},1})_+, (x - x_{\mathrm{k},2})_+, \ldots, (x - x_{\mathrm{k},n})_+\right], \tag{3.35}$$

in which:

$$(x - x_{\mathrm{k},i})_+ = \begin{cases} 0 & \text{if } x \leq x_{\mathrm{k},i} \\ (x - x_{\mathrm{k},i}) & \text{otherwise.} \end{cases} \tag{3.36}$$

The indicator functions are plotted in figure 3.4 as the gray dashed lines. In the same figure, a function that can be constructed with these indicators is given as a solid black line. The kernel function of this indicator function is given by its innerproduct:

$$k(x, y) = \mathbf{f}(x)\mathbf{f}^{\mathrm{T}}(y) = 1 + \sum_{i=1}^{n} (x - x_{\mathrm{k},i})_+ (y - x_{\mathrm{k},i})_+. \tag{3.37}$$

However, the calculation of this innerproduct is computationally demanding, because of the summation. If we want to approximate the relation with an infinite number of splines, the summation is replaced by an integral (Vapnik, 2000).

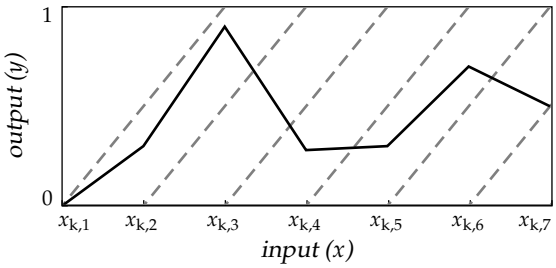
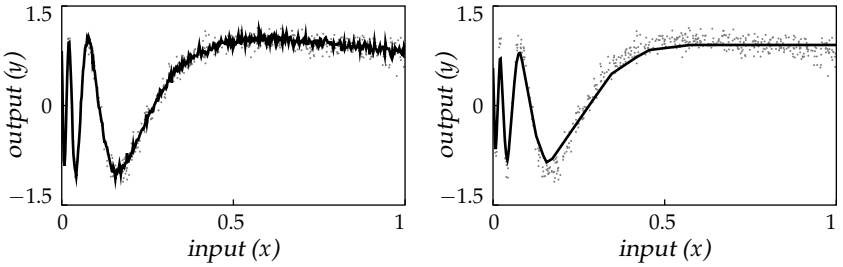


Figure 3.4: Approximation by first order splines. The dashed lines indicate the indicator functions.



(a) $\epsilon = 0.2$ (b) $\epsilon = 0.4$

Figure 3.5: Approximation of function with SVM for different values of ϵ

This leads to the following calculations:

$$\begin{aligned} k(x,y) &= 1 + \int_0^1 (x - x_{k,i})_+ (y - x_{k,i})_+ \, dx_{k,i} && (3.38a) \\ &= 1 + \int_0^{\min(x,y)} (x - x_{k,i})_+ (y - x_{k,i})_+ \, dx_{k,i} && (3.38b) \\ &= 1 + \min(x,y)\,. && (3.38c) \end{aligned}$$

Because of the general set of functions that can be approximated by this kernel, and because of the fast evaluation, this kernel shall be often used in the remaining of this thesis.

With this kernel function the data is approximated. The regularisation parameter C is chosen as $C = 10^5$, which nearly disables the regularisation. The allowable error is set equal to $\epsilon = 0.2$ and 0.4 respectively. The data as well as the approximation are given in figure 3.5. If $\epsilon = 0.2$ there are 311 support vectors and if $\epsilon = 0.4$ only 31 support vectors are required to summarise the data. ∎

In example 3.3 it can be observed that the number of support vectors depends significantly on the insensitivity zone. If this zone is small compared to the noise, a large number of support vectors result, which gives a noisy approximation. On the other hand, if the ϵ zone is large, only a few number of support vectors result, which gives a rough approximation. This can easily be explained by recapitulating that the Lagrangian multipliers are zero within the insensitivity zone.

3.1.4 Least Squares Support Vector Machine

Closely connected to both SVM and dual least squares is the method of Least Squares Support Vector Machine (LSSVM) (Gestel, Suykens, De Moor, and Vandewalle, 2001; Suykens, Van Dooren, De Moor, and Vandewalle, 1999; Suykens and Vandewalle, 2000). The LSSVM method is similar to the SVM setting except that it uses a quadratic criterion instead of an ϵ-insensitive cost function. This makes it nearly equal to the dual least squares setting, the only difference is that LSSVM does not regularise the value of the offset term, see appendix A.1. Furthermore, the solution based on all the training samples (3.22b) is not seen as the final result. Just as with SVM, a subset of these training samples is striven for. A pruning mechanism is used to successively remove those training samples that have little influence on the current approximation, thus obtaining a sparse solution. The sample with the smallest influence on the current approximation is the sample with the smallest parameter.

In De Kruif and De Vries (2003c) an improved pruning mechanism is introduced for LSSVM related to the Optimal Brain Surgery of Hassibi and Stork (1993). This pruning mechanism removes those support vectors that give the smallest increase of errors *after* they are removed. The error that is introduced if support vector i is removed, is:

$$\Delta E_i = \frac{b_i^2}{4[\mathbf{X}^T\mathbf{X}]_{ii}^{-1}}. \tag{3.39}$$

The LSSVM with the alternative pruning mechanism will be indicated as LSSVM+. LSSVM+ gives a smaller approximation error than LSSVM for a fixed number of support vectors, or alternatively, LSSVM+ requires fewer support vectors than LSSVM for an allowable approximation error. This will be illustrated in section 3.5.

Example 3.4

The result of LSSVM before pruning is nearly equal to the result of the dual least squares. This makes the example before pruning applicable to both methods. Data set 1 is again used, while the approximation is done by piecewise linear

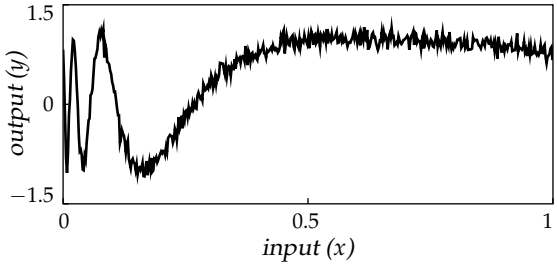
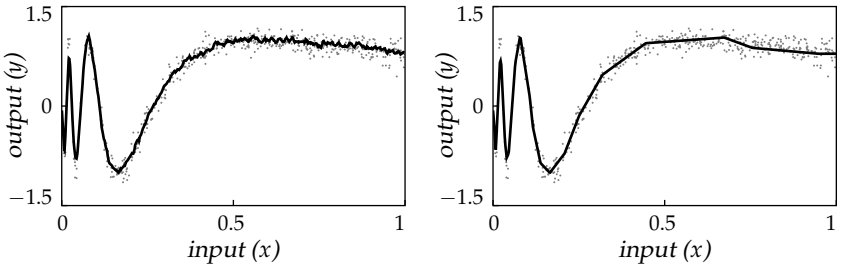


Figure 3.6: Approximation with dual least squares (or LSSVM) without regularisation



(a) Approximation before pruning (b) Approximation after pruning

Figure 3.7: Approximation of data by LSSVM with regularisation

approximation as constructed in example 3.3. First the approximation is made without regularisation. The result is given in figure 3.6. The approximation of the data is noisy which can be explained by looking at the kernel function that is used. This kernel allows all the training samples to be approximated without error, which indeed happens. Incorporating the regularisation term in the LSSVM approximation, results in figure 3.7. In subfigure (a) the result before pruning is given. The result after pruning is given in 3.7(b). Only 20 support vectors remain after pruning. The number of remaining support vectors is user-determined. ▩

3.1.5 B-Spline Network

In previous work on LFFC, Basis-Spline Networks (BSN) were used as function approximator (Velthuis, 2000). A BSN uses a B-spline function as indicator function. The order of the B-spline determines the smoothness. The unscaled spline of order d is given as (Unser, Aldroubi, and

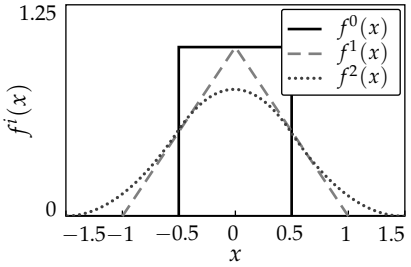


Figure 3.8: Basic splines of different orders

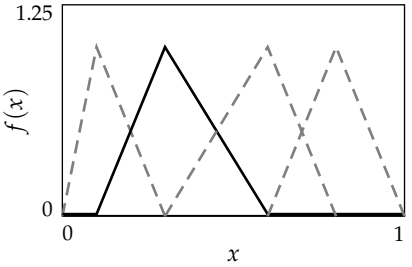


Figure 3.9: A possible BSN built of first order B-spline functions

Eden, 1991):

$$f^d(x) = f^{d-1}(x) \star f^0(x) \quad \text{with} \quad f^0 = 1 \quad \text{iff} \quad -0.5 \leq x < 0.5, \tag{3.40}$$

in which $\star$ indicates the convolution. The basis splines of order zero to two are given in figure 3.8. For multi-dimensional B-splines the tensor product of uni-dimensional B-splines is used (Velthuis, 2000):

$$f^d(x, y) = f^d(x) f^d(y). \tag{3.41}$$

The B-spline functions are distributed on the input space to approximate the data. An one-dimensional example with first-order splines is given in figure 3.9. In this figure it can be clearly seen that the B-spline functions only exert influence on the output in a certain region. This region is called their support. The width of the B-splines can be altered to allow for fast or slowly fluctuating approximations; the symmetry too can be adapted in order to meet the demands on the approximation better.

Due to the limited support of the B-splines, they have to be distributed in such a way that they cover the complete input space. Because the input space is divided into regions by the limited support of the splines, the number of splines required to cover the space grows exponentially with the dimension of this space. The exponential growth of splines is known as the curse of dimensionality and makes it inapplicable to approximate relations with several inputs. With the B-splines seen as indicator functions, the accompanying parameter vector can be calculated as indicated by ordinary least squares.

3.1.6 Radial Base Functions

The Radial Base Function (RBF) is a set of functions that is symmetric around a centre. The Gaussian function that was used in the example with least squares and repeated below is therefore an RBF:

$$f_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\sigma^2}\right). \tag{3.10}$$

An overview of RBF approximation is given in Ghosh and Nag (2001). Just as with B-splines, the RBF can act as an indicator function and can therefore be used in a learning method such as OLS or SVM. If this indicator function is used in a OLS setting, the width, σ, as well as their centres, c_i, should be known to form the indicator matrix $\mathbf{X}$. In the SVM setting only the width has to be known, because the centres are determined by the training data.

The RBF has given rise to a set of learning mechanisms that were especially constructed for this type of function. These mechanisms can alter the number of centres, the location of the centres as well as the widths and are called Radial Base Function Networks (RBFN). A learning mechanism that is capable of changing the centres is given in e.g. (Chen, 1995; Lippmann, 1989). These two references place the centres by means of some vector quantisation algorithm. The result of this approach is that more centres are located in regions where there is more training data. The idea behind it is that if there are more training samples in a region, it is likely that also more predictions are required in this region. Therefore the approximation in this region should be accurate. This does *not* necessarily mean that many centres are required in that region for a correct approximation.

The number of centres that is required for an approximation is altered in e.g. (Chen, Cowan, and Grant, 1991; Fun and Hagan, 1999; Gomm, 2000). They go on adding RBF one by one until some stopping criterion is met. The possible locations of the centres are the input values of the training samples. The location that is selected for a new centre is the location by which the norm of the error after inclusion is minimised. Because these methods add centres, so that the residual is minimised, the centres are located where they are required for a correct approximation. However, selecting one centre at a time does not guarantee the optimal location of the centres (Miller, 1990).

Esposito, Marinaro, and Scarpetta (1998) include centres, as the previous methods do, until a correct approximation is found. However, between two additions the width can be altered for more flexibility to approximate fast fluctuating regions.

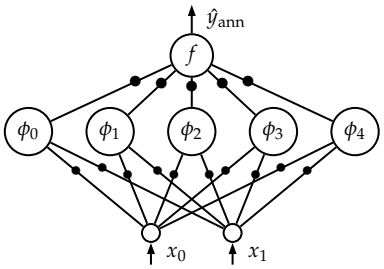


Figure 3.10: The multilayer perceptron

3.1.7 Multilayer Perceptron

A Multilayer Perceptron (MLP) is an Artificial Neural Network (ANN) that consists of a set highly connected small processing units. These processing units typically perform some non-linear function of the weighted sum of the inputs. By altering the weights, different functions can be realised (Haykin, 1994; White, 1992; Zurada, 1992). This thesis only makes use of feedforward ANN with one hidden layer, the MLP, of which the structure is given in figure 3.10. The bottom layer is called the input layer, the middle layer is called the hidden layer and the top layer is called the output layer. In this figure ϕ_i is some non-linear function that is evaluated on the weighted sum of the input variables. The output function f, which is often linear, has the weighted sum of the hidden layer's output for input. The black dots in the figure represent the weights. The output is:

$$\hat{y}_{\text{ann}} = f\left(\sum_{i=1}^{n} w_i \phi_i \left(\sum_{j=1}^{k} w_{ij} x_j\right)\right), \tag{3.42}$$

in which w_i is the weight connecting the output neuron with hidden neuron i, and w_{ij} is the weight connecting input j with hidden neuron i. If the term neural network is used in this thesis, it always concerns a multilayer perceptron.

The weights will get a value during the training phase. The data set is repeatedly supplied to the MLP and the weights are adapted in the direction which decreases the error. This is done by the back propagation algorithm that can be found in (Haykin, 1994; White, 1992; Zurada, 1992). This optimisation routine often gets stuck in a local minimum.

The number of hidden neurons determines the freedom the network has to approximate the relation within the data. The number of hidden neurons is unknown beforehand and can be determined by trial and

error. Just as with RBF functions, the number of hidden neurons can be increased during training (Fahlman and Lebiere, 1990), or the number of hidden neurons can be decreased after learning (Cun, Denker, and Solla, 1990; Hassibi and Stork, 1993; Prechelt, 1997; Reed, 1993; Stahlberger and Riedmiller, 1997; Van de Laar and Heskes, 1999).

3.2 Discussion

Although it would be convenient if one of the above mentioned methods could be directly used in a control setting, it is believed to be impossible due to their idiosyncracies. However, investigating these peculiarities will give a possibility to combine their potential to obtain a function approximator with advantageous properties.

3.2.1 Ordinary Least Squares

The OLS method is well documented and therefore, much is known about its properties. See, e.g. (Björck, 1996; Draper and Smith, 1998; Golub and Van Loan, 1996; Kleinbaum et al., 1987; Stewart, 1998). Due to the close resemblance with projections, algorithms taken from linear algebra can be used to support the implementation of OLS. The OLS method results in an unbiased, minimal variance estimate.

The main difficulty with OLS is to select a good set of indicator functions beforehand.

3.2.2 Support Vector Machines

The SVM uses only a limited amount of the training samples for the prediction of the data. The calculations for the prediction and the calculations to get the parameters needed for the prediction, are done based on the innerproducts in the feature space. An innerproduct is an (unnormalised) similarity measure and the SVM therefore uses the similarity of the new sample with the training samples in some space for prediction. Because the prediction is based on similarities with training samples, there is no need to divide the input space into regions, which makes it less prone to the curse of dimensionality. This makes the sample-based approach attractive to use for function approximation in learning control.

However, the use of the ϵ-insensitive cost function gives rise to undesirable behaviour. First, which was already illustrated in the section on SVM in figure 3.5, there is the sensitivity to the choice of the value for ϵ. If the value of ϵ is selected too small, too large a set of support

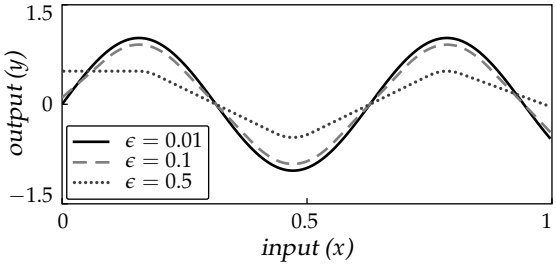


Figure 3.11: SVM-approximation of a sine with $\epsilon = 0.01, 0.1, 0.5$ respectively

vectors will result, giving a noisy approximation. On the other hand, if the value of ϵ is too large, the extrema will be cut off. This behaviour is illustrated in figure 3.11 and is accounted for by the insensitive zone. The number of the indicators used, is respectively 507, 384 and 119 for $\epsilon = 0.01, 0.1$ and 0.5. The regularisation parameter C was given a value of 10^6.

Second, the number of support vectors is rather large for the complexity of the relation in the approximation of figure 3.11. For the approximation in which an error of 0.5 is tolerated, the number of required support vectors is 119. All these support vectors are located at the extrema of the sine. This behaviour can be explained if the regularisation term is observed. It minimises the 2-norm of the weight vector. The value of this term will be smaller if more small parameters are used, instead of one large parameter. This results in locally concentrated support vectors.

The third inconvenience is the oscillatory approximation if two training samples have a nearly equal input, but a difference in output larger than 2ϵ. As a sample with a prediction error larger than ϵ becomes a support vector, both samples become support vectors and try to minimise their own approximation error. The approximation resulting is therefore oscillatory. See section 3.5.1 for an example of this behaviour.

So, although the sample-based approach is attractive, the disadvantages due to the ϵ-insensitive zone make the SVM unattractive for direct use in learning control.

3.2.3 Least Squares Support Vector Machine

The use of the quadratic cost function avoids the disadvantages in SVM due to the ϵ-insensitivity zone. However, the sparseness that was inherently due to this insensitivity zone is no longer there. Sparseness has

to be arrived at by pruning. When significantly fewer support vectors are required than there are training samples, the pruning will take considerable computation time. The result represents, just as in SVM, the complete data set with only a limited number of samples.

The pruning mechanism for LSSVM omits those samples from the training set that do not largely influence the prediction. However, omission of samples from the training set removes information on the underlying relation, which is an undesired property.

Furthermore, LSSVM has difficulties with double input samples. If there are two equal inputs, the matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ will be singular, making it impossible to determine the vector $\boldsymbol{\alpha}$ of (3.19). The regularisation term will make the system solvable, although the numerical condition of the system will be bad for small regularisation.

3.2.4 Multilayer Perceptron

As already mentioned in section 3.1.7, neural networks often get stuck in local minimum. Besides, the number of neurons to approximate the data is unknown beforehand. What has not been mentioned yet, is that it is a time-consuming task to train an MLP. All the data needs to be repeatedly presented to the network before the parameters converge.

After convergence of the parameters to some minimum, superfluous weights can be pruned (Cun et al., 1990; Hassibi and Stork, 1993; Prechelt, 1997; Reed, 1993; Stahlberger and Riedmiller, 1997; Van de Laar and Heskes, 1999), but this will increase the training time. In addition, the superfluous weights are pruned for the found minima, which may be local and therefore not useful anyway.

The idea of Fahlman and Lebiere (1990) is to include one neuron a time. The input weights of the newly added neuron are trained in such a way that it has the largest correlation with the remaining error, and are then fixed. Afterwards, the output weights of all the neurons are retrained. If the remaining error is still too large, a new neuron is included. The decision when to stop this procedure is left to the user.

This idea is closely related to the forward subset selection scheme given in Miller (1990). In this scheme a basis function is chosen from a set of possible indicator functions that is most closely correlated to the remaining approximation error. The difference is that Miller uses a discrete set of indicator functions, while Fahlman and Lebiere use a potential indicator function that has some parameters that are tuned until it is as close as possible to the residual. The advantage of the growing neural network approach over static neural networks is that the number of neurons is fit for the task on hand.

3.2.5 Radial Base Function Networks

Just as the B-spline, the RBF is a function that can be used as an indicator function, and the influence of this function is semi-local. This makes the indicator function prone to the curse of dimensionality if it is set with fixed widths and centres.

The methods that include centres with a fixed width at locations of input samples spans the same set of functions as the (LS)SVM. Because the width is fixed, the space is divided into regions, although the centres of these regions are undetermined.

In Esposito et al. (1998) the RBFN grows with one basis function at a time. The new RBF is centred at the location where there is the largest residual. The RBF is only active in a limited region which is determined by the location of the other centres. Because it alters the spread of the function and the region of activation based on the data, it is less likely to fall for the curse than when fixed spread is used.

3.3 Proposition of Key Sample Machine

Based on the evaluation of the different methods, we believe that a new approximator can be constructed that better suits our needs by combining the following ideas:

- Use the summed squared approximation error as *cost function*,
- Use the *sample-based* approach of SVM and LSSVM,
- Use some of the samples for the prediction, but use all the samples for the training (*data compression*),
- Use a s*ubset selection scheme* that fits the problem on hand to select the samples that are needed for the prediction.

These points are treated separately. An overview of the final algorithm is given in algorithm 3.1 on page 59.

3.3.1 Cost function

A quadratic cost function is chosen for the following reasons:

Literature: This cost function is used frequently and therefore there exists a vast amount of literature.

Unbiased: The use of the quadratic cost function gives an unbiased approximation. This in contrast with the ϵ-insensitivity function that results in a biased estimate (Chan, Chan, Cheung, and Harris, 2001).

ML for Gaussian noise: The least squares cost function results in a maximum likelihood estimate of the approximation for Gaussian noise. Gaussian noise is often assumed although this makes the method sensitive to outliers.

Implementation: Because the close resemblance between the solution of the least squares problem and projections in the field of linear algebra, there is a vast amount of algorithms to support the implementation.

The selection of the appropriate training samples now becomes a separate subproblem, while for SVM the selection of the samples was inherently present in the optimisation problem. This separation opens the possibility to use a selection mechanism that best fits the problem on hand.

3.3.2 Sample-based

The methods that use the innerproduct to train the parameters for the prediction, as well as for the prediction itself, will be called *sample-based,* because its calculations are based on similarities with the training samples. In such schemes the input space is not necessarily divided into regions and an exponential growth of the number of parameters with increasing input dimension can be circumvented. The curse of dimensionality limited the performance of the learning control scheme in previous work. Therefore, to be less prone to the curse of dimensionality, we use a sample-based approach.

The actual number of samples that summarises the data depends on the complexity of the underlying relation in the data and of the feature space that is used. Adapting this number is equivalent to using a flexible number of hidden neurons or RBFs.

3.3.3 Data compression

As with LSSVM(+), we want to decrease the number of samples required to *calculate* a prediction for a new sample, but opposed to LSSVM(+), we do *not* want to decrease the number of samples for the training. So, for

the prediction in (3.22b) and repeated below

$$\hat{y}_{\mathrm{d}}(\mathbf{x}_{\mathrm{new}}) = \sum_{i=1}^{N} \langle \mathbf{f}(\mathbf{x}_{\mathrm{new}}) \, , \mathbf{f}(\mathbf{x}_i) \rangle \alpha_i, \tag{3.22b}$$

we do not want to take the weighted sum with the innerproducts for all the N training samples for a prediction, but we do want to use N training samples to calculate α_i.

In order to use all the samples for training, and only a limited number for prediction, we use the kernel function with a set of selected samples, the *key samples*, as indicator functions, and an OLS scheme to calculate the corresponding $\boldsymbol{\alpha}$. Later, it will be explained how this limited set of key samples is found.

First, it is shown how the kernel function for a sample can be interpreted as an indicator function for the OLS scheme. For sample i, with input $\mathbf{x}_i$, we define the *dual indicator function* $\tilde{f}_i(\mathbf{x})$ as:

$$\tilde{f}_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i) = \mathbf{f}(\mathbf{x}) \mathbf{f}(\mathbf{x}_i)^{\mathrm{T}}. \tag{3.43}$$

The second equality was treated in section 3.1.3. Substituting the dual indicator function (3.43) into the prediction for dual least squares (3.22b) results in:

$$\hat{y}_{\mathrm{d}}(\mathbf{x}_{\mathrm{new}}) = \sum_{i=1}^{N} \tilde{f}_i(\mathbf{x}_{\mathrm{new}}) \alpha_i \tag{3.44}$$

Comparing this with the predication for the ordinary east squares (3.8)

$$\hat{y}_{\mathrm{ls}}(\mathbf{x}_{\mathrm{new}}) = \sum_{i=1}^{n} f_i(\mathbf{x}_{\mathrm{new}}) b_i \tag{3.45}$$

clearly shows the resemblance between these approaches. Now, if only n key samples are used for the prediction and there are N samples in the training set ($n \leq N$), then, similar to OLS, an indicator matrix can be constructed:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{f}_1(\mathbf{x}_1) & \tilde{f}_2(\mathbf{x}_1) & \cdots & \tilde{f}_n(\mathbf{x}_1) \\ \tilde{f}_1(\mathbf{x}_2) & \tilde{f}_2(\mathbf{x}_2) & \cdots & \tilde{f}_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_N) & \tilde{f}_2(\mathbf{x}_N) & \cdots & \tilde{f}_n(\mathbf{x}_N) \end{bmatrix} \tag{3.46a}$$

$$= \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_{\mathrm{ks},1}) & k(\mathbf{x}_1, \mathbf{x}_{\mathrm{ks},2}) & \cdots & k(\mathbf{x}_1, \mathbf{x}_{\mathrm{ks},n}) \\ k(\mathbf{x}_2, \mathbf{x}_{\mathrm{ks},1}) & k(\mathbf{x}_2, \mathbf{x}_{\mathrm{ks},2}) & \cdots & k(\mathbf{x}_2, \mathbf{x}_{\mathrm{ks},n}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_{\mathrm{ks},1}) & k(\mathbf{x}_N, \mathbf{x}_{\mathrm{ks},2}) & \cdots & k(\mathbf{x}_N, \mathbf{x}_{\mathrm{ks},n}) \end{bmatrix}, \tag{3.46b}$$

in which $\mathbf{x}_{\text{ks},i}$ denotes the i^{th} key sample. Using a target vector and a parameter vector identical to (3.2), then the normal equations (3.7) can be used to calculate the parameter vector $\mathbf{b}$:

$$(\tilde{\mathbf{X}}^{\text{T}}\tilde{\mathbf{X}})\mathbf{b} = \tilde{\mathbf{X}}^{\text{T}}\mathbf{y}. \tag{3.47}$$

The prediction for a new input $\mathbf{x}_{\text{new}}$ is calculated as:

$$\hat{y}(\mathbf{x}_{\text{new}}) = \sum_{i=1}^{n} \tilde{f}_i(\mathbf{x}_{\text{new}})b_i = \sum_{i=1}^{n} \langle \mathbf{f}(\mathbf{x}_{\text{new}}) , \mathbf{f}(\mathbf{x}_i) \rangle b_i. \tag{3.48}$$

in which only n key samples are used for the prediction. However, for the calculation of the parameters b_i, N samples are used.

With these sample-based dual indicator functions, only a limited set of samples is used to predict all the data, but all the data is used to train the parameters corresponding to the limited set of indicator functions.

3.3.4 Subset selection scheme

With the dual indicator functions used in an OLS scheme, the parameter values can be calculated. However, the dual indicator functions for prediction, still have to be selected. This is equal to the selection of an appropriate structure for the approximation. Because the dual indicator functions are directly related to the training samples (3.43), selecting a set of dual indicator functions is the same as selecting a set of key samples from the training samples. All the training samples are considered potential key samples, or, in a different terminology, all the dual indicators function stemming from the training samples are potential indicator functions.

There are several methods available for the selection of indicators from a set of potential indicators (Chen et al., 1991; Draper and Smith, 1998; Funival and Wildon Jr, 1974; Miller, 1990). These methods can be divided into three main groups:

1) forward selection,

2) backward elimination,

3) complete search.

The forward selection starts without indicators and will add *that* indicator that will minimise the sum squared error or some other norm. The backward elimination will start with all the possible indicators and will omit one indicator at a time until some criterion no longer holds. The

complete search will test all the possible combinations of indicators. This can be done in such a way that not all the combinations are actually tried (Funival and Wildon Jr, 1974), but the computational time is still prolonged. The first two methods cannot guarantee finding the optimal solution (Miller, 1990). It goes without saying that combinations of these methods are possible.

As the number of required indicators is generally much smaller than the number of potential indicators, forward selection is generally much faster than backward elimination, and for this reason the forward selection method is used hereafter. The testing of all the combinations is not a feasible method when the number of potential indicators is significant.

The forward selection scheme adds one indicator at a time. The potential indicator function that is added is that indicator that maximally decreases the remaining summed squared approximation error after inclusion.

It follows from (3.46b) that each indicator generates a column vector in $\mathbf{X}$. The height of this column vector corresponds with the number of samples. The residual of the current approximator for the given training set is given as $\mathbf{e} = \mathbf{y} - \mathbf{X}\boldsymbol{\alpha}$, which is also a vector of dimension N. Now we want to add the indicator, that decreases the summed squared approximation error most. If an indicator could be found that is linearly related to the residual vector $\mathbf{e}$, the inclusion of that indicator would make the residual zero. The presence of such an indicator is not likely. However, its closest match is that potential indicator vector that will generate a column vector in $\mathbf{X}$ that makes the smallest absolute angle with the residual vector $\mathbf{e}$. Or more precisely, the maximum of the cosine squared of this angle:

$$\mathrm{err}_i = \cos(\theta)^2 = \frac{\left(\mathbf{e}^{\mathrm{T}}\mathbf{a}_i\right)^2}{\mathbf{a}_i^{\mathrm{T}}\mathbf{a}_i \left(\mathbf{e}^{\mathrm{T}}\mathbf{e}\right)}, \tag{3.49}$$

in which err_i is the normalised change in error due to the inclusion of potential indicator i and $\mathbf{a}_i$ the column vector spanned by this indicator for all the samples. The potential indicator with the largest change in error should be selected for inclusion. This result has been proven to maximise the decrease of the summed squared approximation error (Chen et al., 1991; Fun and Hagan, 1999; Miller, 1990). As argued in section 3.2.4, this selection scheme is closely related to the selection scheme used in (Fahlman and Lebiere, 1990).

Example 3.5 (Successive steps of an approximation)
We want to approximate 25 data points of data set 1 by first-order B-splines. For

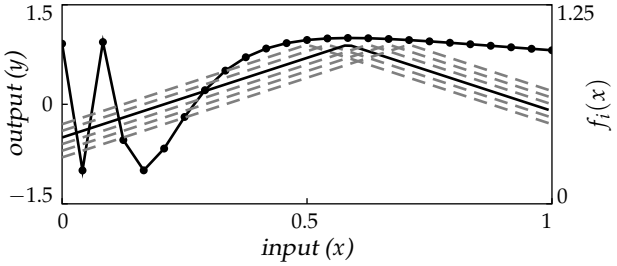


Figure 3.12: Six potential indicator functions

this example, the data points are distributed at equal distances across the input range. The potential indicator functions for this approximation are:

$$f_i^1(x - x_i) = f^0(x - x_i) \star f^0(x - x_i), \quad i = 1 \ldots 25 \tag{3.50}$$

with

$$f^0 = \begin{cases} 1 & \text{if } -0.5 \leq x - x_i < 0.5, \\ 0 & \text{else.} \end{cases} \tag{3.51}$$

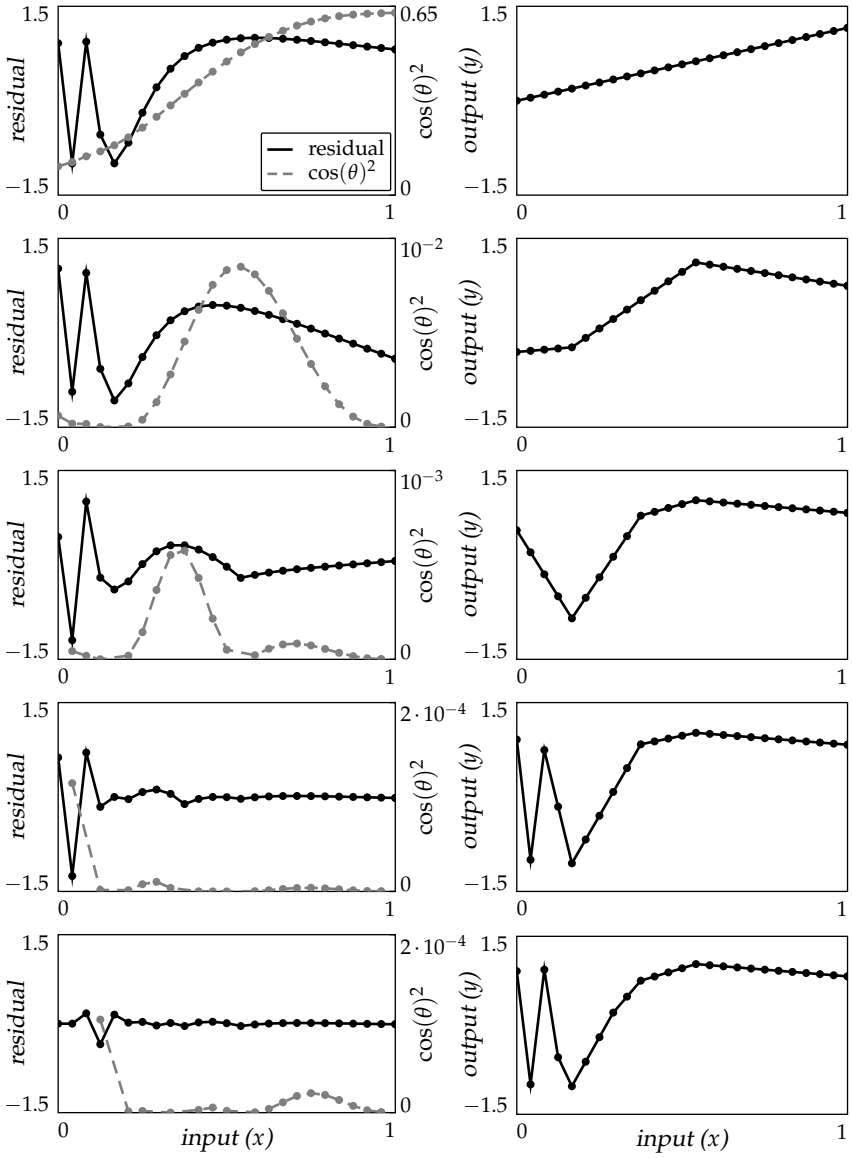
Six of these potential indicator functions are shown in gray in figure 3.12, one of which is highlighted. These potential indicators differ in the location of their centres. In this figure the function we want to approximate is also shown.

In figure 3.13, successive steps of the approximation are given. The top-left figure shows the residual in black and the cosine squared of the angle between the residual and the potential indicator function in gray. This cosine squared gives the correspondence of the potential indicator and the residual. As there is no prediction in the first step, the residual is equal to the targets. It can be seen that the maximal value of the correspondence occurs at $x = 1.0$ and hence this sample is the first indicator that is selected. The prediction with this one key sample is given in the top-right.

The residual after two inclusions is given in the second graph from the top on the left. The correspondence is plotted in the same figure. After the inclusion of the new key sample, the approximation is shown on the right-hand side of this figure. The figures show the approximation after inclusion of the first, third, fifth, seventh and ninth key sample.▩

Stopping criteria

When the best potential indicator has been identified, it should be decided whether it is good enough to be included for the prediction or that the inclusion of more indicators should stop. A criterion is proposed that keeps including more indicator as long as the probability



(a) Residual and correlation. From top to bottom: After 0, 2, 4, 6 and 8 inclusions

(b) Approximation, from top to bottom: After 1, 3, 5, 7 and 9 inclusions

Figure 3.13: Successive steps of the approximation

is too small that the new parameter is zero. Hence, this test evaluates whether noise is fitted or the underlying relation.

Define the extra sum of squares S^2 as the difference between the summed squared approximation errors of all samples before and after the inclusion of the candidate indicator. In (Draper and Smith, 1998; Kleinbaum et al., 1987) it is shown that for additive Gaussian noise on y, S^2/σ^2 is distributed as χ^2_1 if and only if the new indicator's parameter is zero:

$$\frac{S^2}{\sigma^2} \sim \chi^2_1 \quad \Leftrightarrow \quad \alpha_i = 0. \tag{3.52}$$

In this equation, σ^2 is the variance of the noise.

The hypothesis '$\alpha = 0$' is tested, and if this hypothesis is rejected with some predefined significance, the candidate indicator becomes an indicator. If the new parameter is zero, then the probability that a certain error reduction is found is calculated as:

$$P\left(\frac{S^2}{\sigma^2} \geq z_\zeta \middle| \alpha = 0\right) < \zeta. \tag{3.53}$$

In this equation ζ denotes the probability that a realisation of z_ζ or larger is found. ζ is called the *significance level* and the corresponding value of z_ζ follows from the χ^2_1 distribution. If the probability for the found error reduction is too small, then the new parameter is unlikely to be zero, and therefore, the indicator is included.

Example 3.6 (Stopping criterion)
For example, the summed squared approximation error changes from 11 to 10. The variance of the noise in this example is 0.25. This yields:

$$P\left(\frac{1}{0.25} \geq 4 \middle| \alpha = 0\right) < 4.6\%. \tag{3.54}$$

The probability that this error reduction is found, is smaller than 4.6% if $\alpha = 0$. If the significance level of the rejection is set at 5%, the hypothesis $\alpha = 0$ is rejected, and therefore, the indicator is included. ▩

The noise variance in (3.53) is used to determine when the approximator has to stop including more basis functions. As long as the ratio S^2/σ^2 is larger than z_ζ, the inclusion of new indicators continues. The noise variance is often unknown for real-life problems and has to be estimated. However, the noise estimate can also be interpreted as a *design parameter:* setting it large will give a rough approximation, because the inclusion is stopped early. For a small estimate of the noise level, the

Algorithm 3.1: Algorithm for Key Sample Machine

Given: Training set $(\mathbf{x}_i, y_i)_{i=1...N}$
Initial decisions: Noise estimate σ and kernel function
1: form potential indicators $\tilde{\mathbf{f}}_i$ based on $\mathbf{x}_i$ and kernel for all i (3.43)
2: find potential indicator with largest correlation to current residual (3.49)
3: **while** inclusion is significant (3.53) **do**
4: add this potential indicator to the set of indicators (3.46)
5: find potential indicator with largest correlation to current residual (3.49)
6: **end while**

approximation will be accurate. Care should be taken that, if the parameter is set *too* small, the KSM will start fitting noise.

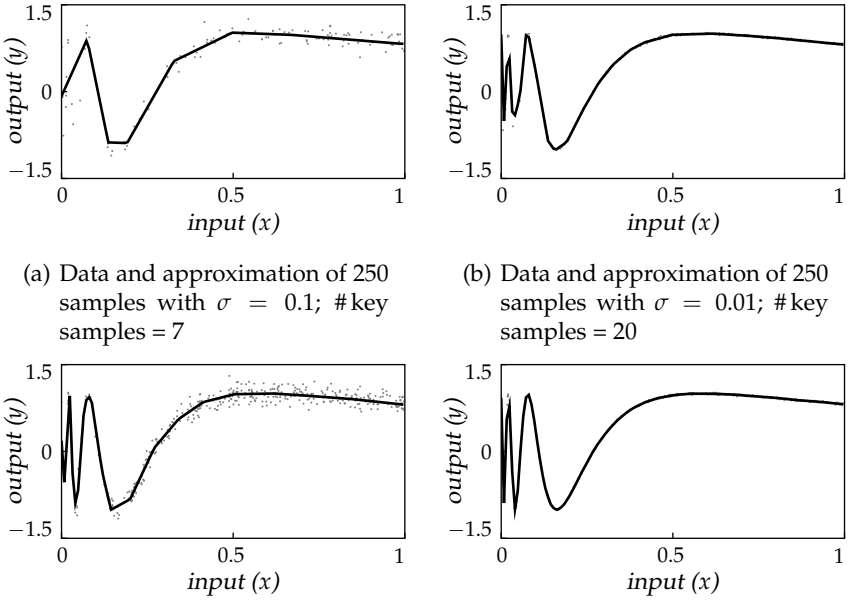
The significance level ζ is not considered an extra parameter that has to be given a value, because the effect it has on the inclusion of a sample, can also be obtained by altering the noise level. For a given significance level, the noise estimate σ can freely alter the value for which the hypothesis is rejected. It is therefore proposed to select a significance level once, and use the noise estimate as a design parameter to indicate the accuracy by which the data should be approximated.

3.3.5 Overview of the algorithm

The introduced approximator will be referred to as the *Key Sample Machine* (KSM). The complete algorithm for KSM is given in algorithm 3.1. Before the approximation starts, a kernel function has to be selected to form the dual indicator functions. The noise level estimate acts as a design parameter and should also be assigned a value. If a true noise level is known, this can be selected. However, then the significance level should be set to a meaningful level.

The test if the inclusion is still significant is done by (3.53). The selection of a potential indicator that comes closest to the residual is calculated, based on the angle between the residual and the indicators as treated in section 3.3.4. An implementation for this algorithm is treated in appendix B.1.

Different subset selection schemes available in literature can be used instead of the forward selection scheme (Miller, 1990).



(a) Data and approximation of 250 samples with $\sigma = 0.1$; # key samples = 7

(b) Data and approximation of 250 samples with $\sigma = 0.01$; # key samples = 20

(c) Data and approximation of 2000 samples with $\sigma = 0.1$; # key samples = 17

(d) Data and approximation of 2000 samples with $\sigma = 0.01$; # key samples = 48

Figure 3.14: Approximation of data with different noise levels and number of training samples

The given algorithm can easily be adapted to incorporate a priori knowledge in the form of indicator functions. These a priori indicator functions can be interpreted as a set of potential indicators that has already been included. Additional indicators from the set of potential indicators are selected as before.

3.4 Evaluation

A series of function approximations is made to illustrate and test the performance of the KSM. The first set of function approximations is performed in order to test whether the stopping criterion works appropriately.

Stopping criterion

The results of these approximations are illustrated in figure 3.14(a) up to (d). Data from data set 1 is used with a different number of training samples and noise. The number of training samples that is used, is respectively 250 or 2 000 and the noise standard deviation is $\sigma = 0.1$ or $\sigma = 0.01$. The set of potential indicator functions was chosen as $f_i(x) = 1 + \min(x, x_i)$. This results in a piecewise linear approximation. No indicator functions were included as a priori knowledge. The only parameter that has to be given a value, is the estimate of the noise variance for the use in (3.53). For this set of evaluations, it is selected as the correct noise level. The sensitivity of the approximation for this parameter will be shown later.

Figures 3.14(a)-(d) show clearly that the approximation of the data will be rough when there is little (a, b) or noisy data (a, c), while the approximation is more accurate, when good (b, d) or much data (c, d) is available. This is exactly what is expected from the stopping criterion.

Parameter sensitivity

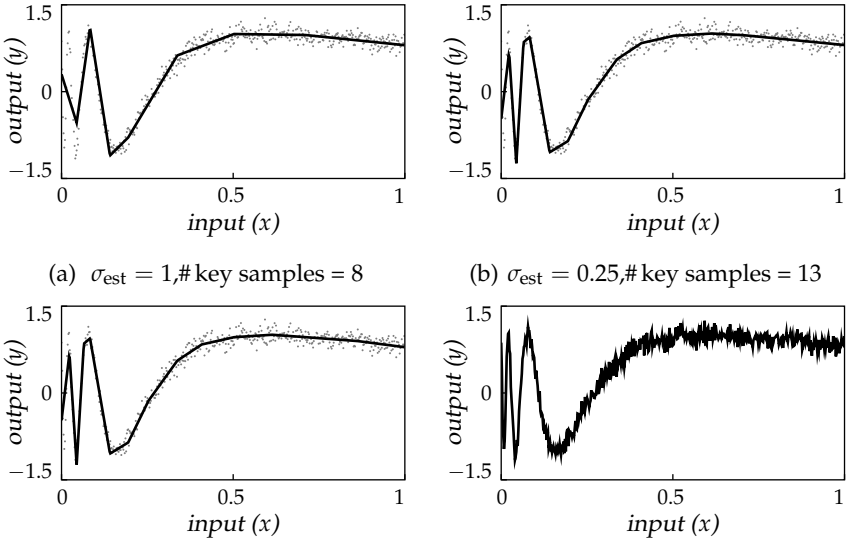
The noise variance is used to determine when the approximator has to stop adding more basis functions and can thus be used as a design parameter. The effect of different noise estimates is shown in figure 3.15. The behaviour of the approximator is just as expected. The number of key samples to approximate the relation increases from 8 via 13 and 14 to 832. Clearly, in the last case the method is fitting noise.

Dual targets

One of the motivations to construct a function approximator, was that (LS)SVM had difficulties with samples with equal inputs but with different outputs, see section 3.2. The behaviour concerning identical input samples is evaluated by the use of data set 2. The results are shown in figure 3.16. Inspections of the results show a good approximation. Depending on the estimated noise level, fluctuations that occur between 0.12 and 0.15 [m] are approximated or not. This shows the use of the noise estimate as a design parameter.

Selected structure

The purpose of the last set of approximations is to test whether the subset selection scheme adds the correct key samples so that an appropriate



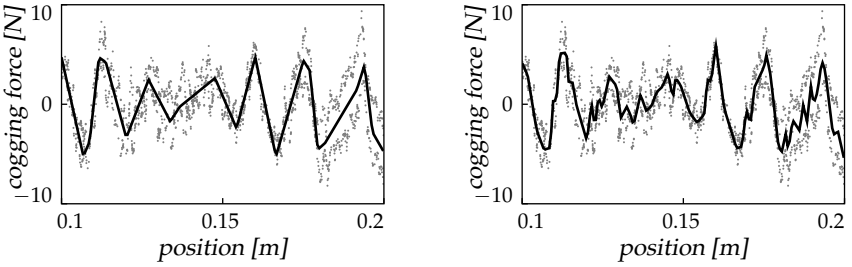
(a) $\sigma_{\text{est}} = 1$,# key samples = 8

(b) $\sigma_{\text{est}} = 0.25$,# key samples = 13

(c) $\sigma_{\text{est}} = 0.1$ (correct),# key samples = 15

(d) $\sigma_{\text{est}} = 0.01$,# key samples = 959

Figure 3.15: Parameter sensitivity; approximation of data with different noise estimates



(a) $\sigma_{\text{est}} = 5$

(b) $\sigma_{\text{est}} = 2$

Figure 3.16: Approximation of data set 2

structure for the data is found. Data set 1 is used for the function approximation with a variable number of training samples. The results are the average of five runs. The noise level is kept constant at $\sigma = 0.05$.

In these approximations, we are interested in the structure only, not in the parameters. Therefore, after completing the approximation, the structure found is kept unchanged, and the parameters are retrained

Table 3.1: Mean squared error for different numbers of samples

data size	**MSE**		**nr. of indicators**		**MSE** $_{\text{opt}}$
	mean	std	mean	std	
100	$3.77 \cdot 10^{-2}$	$2.17 \cdot 10^{-2}$	10.8	1.9	$0.101 \cdot 10^{-2}$
1 000	$1.68 \cdot 10^{-3}$	$1.30 \cdot 10^{-3}$	19.6	1.5	$0.376 \cdot 10^{-3}$
10 000	$4.28 \cdot 10^{-4}$	$3.51 \cdot 10^{-4}$	25.6	3.1	$1.81 \cdot 10^{-4}$

with a large noiseless data set using OLS. This results in the best possible approximation of the relation *for the structure found.*

The Mean Squared Errors (MSE) of the approximations , as well as the standard deviation of it, are given in table 3.1. Also the average number of key samples and their standard deviations are given. The last column gives the smallest MSE that can be found for the average number of key samples. This optimum value is calculated by Matlab optimisation toolbox. So, the optimal MSE for 10.8 key samples is $0.101 \cdot 10^{-2}$. This value is a weighted average of the optimal value for 10 and 11 key samples.

Based on the figures in the table, one can see that the selected structure can approximate the underlying relation better if more samples are supplied to it. This is fairly obvious, because the method will only add a new indicator if it is statistically relevant. If more data is available, more indicators will be statistically relevant.

In the table one can also see that the difference between the approximator and Matlab's optimisation routine decreases if the number of samples increases. So, it not only adds *more* key samples, it also adds *better* key samples. There are two reasons for this: the possible structures of the learning mechanism are based on the samples; because there are more samples, the number of possible structures increases. Second, because more data is available, a better test can be made to see whether a correct term is added to the structure.

3.5 Comparison

The different methods that have been discussed in this chapter will be compared with each other to test their performance. This will be done in two phases: first the approximators will be compared that are based on the SVM methodology. All these methods use a limited set of samples to summarise the data, and therefore a fair comparison can be made.

The comparison will focus on the efficiency of data compression, the capability of approximating noisy data and the possibility to handle dual targets.

Next, a set of experiments is performed to test the applicability of the approximators on high dimensional input. Finally, computation time and ease of use are compared.

3.5.1 Support-vector based methods

Data compression

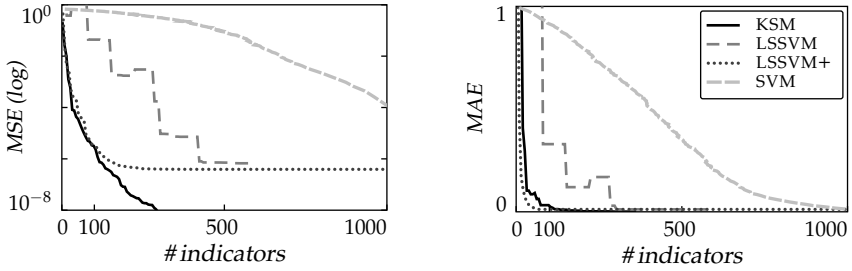
The first comparison tests the errors that result by summarising all data with only a limited set indicators. Indicators can refer to support vectors or key samples for the remaining thesis. 2 000 training samples from the first data set are fed noiseless to KSM, LSSVM, LSSVM+ and SVM. The error is evaluated while growing or pruning the network. All the methods approximate the relation by a piecewise linear relation. Noiseless data is used, implying that only the performance on data compression is investigated.

In figure 3.17(a) the MSE of the different methods is given as a function of the number of indicator functions, while in figure 3.17(b) the Maximal Absolute Error (MAE) is shown. The LSSVM+ and KSM outperform the SVM and LSSVM by far. If, e.g. an MSE of 10^{-3} is allowed, KSM needs 22 key samples, LSSVM+ keeps 26 support vectors, LSSVM requires 283 while SVM uses a total of 874 support vectors.

The LSSVM+ and KSM have a comparable MSE if only few indicators are used; for a larger number of key samples, the KSM is much better. The maximal absolute error for few support vectors is smaller for the LSSVM+. These differences can be accounted for, because the LSSVM+ prunes those samples that will minimally increase the MAE, while KSM includes the indicator that minimises the MSE.

The decrease of the error for LSSVM and LSSVM+ does not continue above a certain number of support vectors. One would expect that if there is no noise, more indicators will give a better approximation result. Investigation of the error shows that errors occur when two input samples are located near each other. These similar samples will give similar indicator vectors, resulting in near linear dependence in the indicator matrix. This, in its turn, results in a bad numerical condition for solving the parameter vector.

Commenting on the pruning mechanism of the LSSVM variants, we can see that the pruning in LSSVM+ results in a significant diminished



(a) Mean Squared Error

(b) Maximal Absolute Error

Figure 3.17: Data compression performance: the approximation error for different methods as a function of the number of indicators

approximation error. However, the cost for this is the increased computational load.

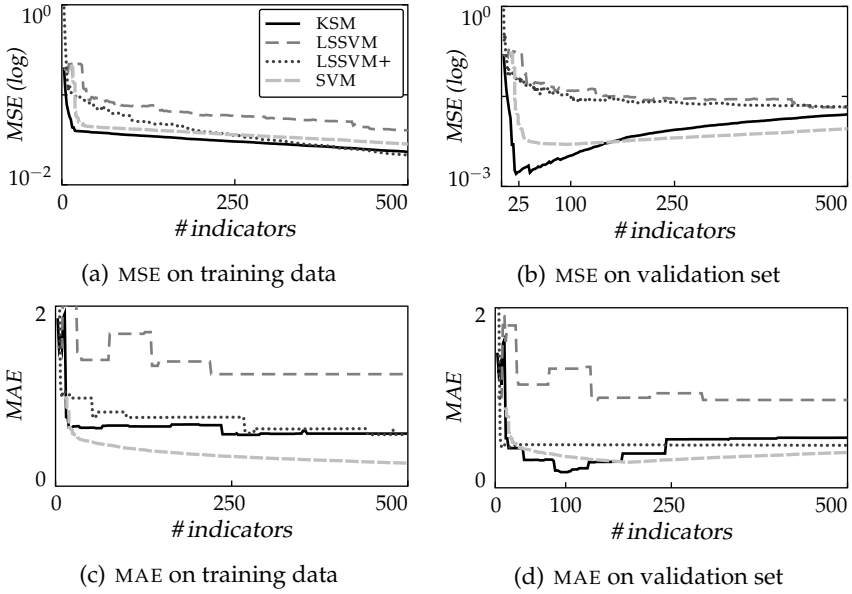
Note, that in general, the most relevant part of figure 3.17 is the area with only a limited number of indicators; between 1 and 100 as regards this example. These approximations do not require a lot of memory and can cope with noise.

Noise handling

The second comparison will treat the noise handling ability of the methods. Noise corrupted data of data set 1 is used for the training, while the noiseless function is used for validation purposes. The results of the approximations are given in figure 3.18. In subfigure (b) it can be seen that the KSM gives the best approximation on the validation set. Moreover, this small error is achieved with fewer key samples than the other methods need for their minimum. So, KSM gives the smallest error for the smallest number of indicators in this example. When the number of indicators becomes large, KSM starts fitting noise.

If the statistical stopping criterion is used for KSM, the number of indicators is found to be 25. This is indicated in figure 3.18(b) and is near the minimal validation MSE that is found at 20 indicators. When more key samples are included, the MSE grows, but the MAE decreases until 93 key samples are included.

The MSE on the validation set of LSSVM with both methods of pruning is large, compared with the other methods, while the MAE of LSSVM+ remains small.



(a) MSE on training data

(b) MSE on validation set

(c) MAE on training data

(d) MAE on validation set

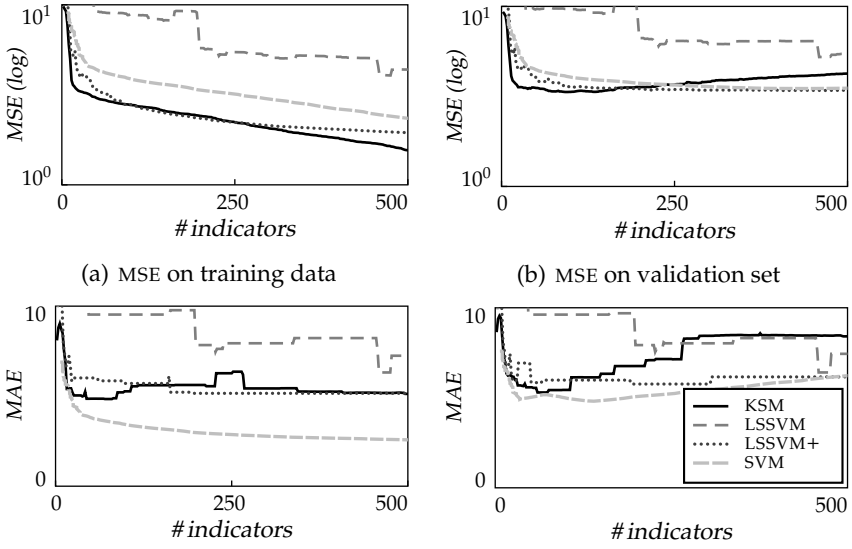
Figure 3.18: Error on the approximation of data set 1 corrupted with noise as a function of the number of indicators.

An interesting observation is that the MSE on the validation set of SVM is smaller if there is noise corrupting the targets. This can be explained because the noise will form a 'band' around the function that has to be approximated. Because of this band, the insensitivity zone will not make the approximation to cut the extrema, but makes it approximately coincide with the underlying function.

Double targets

The last in this set of experiments is the approximation of data set 2. 2 000 randomly selected samples of this data set are used for validation purposes. The result of the approximation is given in figure 3.19. Again KSM gives the smallest MSE on the validation set for the smallest number of indicators. However, if more key samples are included into the approximation, the KSM starts to fit noise. In this approximation the SVM gives the best results on the validation set measured in MAE.

The validation error of the SVM remains relatively small even for a large number of indicators. Inspection of the approximation shows that it fluctuates swiftly between the outer values of the insensitivity band.



(a) MSE on training data

(b) MSE on validation set

(c) MAE on training data

(d) MAE on validation set

Figure 3.19: Error on the approximation for data set 2

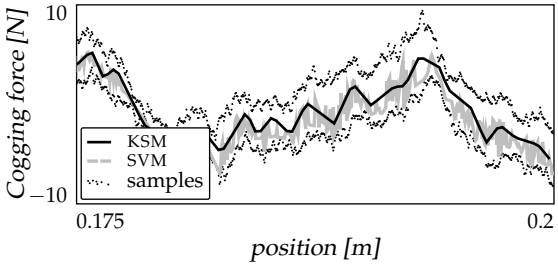


Figure 3.20: Data with ambiguous samples and the approximation hereof by SVM and KSM

This behaviour is shown in figure 3.20. The approximation is given for the smallest MSE on the validation set. Due to the double targets, there is always a large error, making this fluctuation behaviour not clear in the MSE of the validation set. This behaviour can be explained by noting that all training samples outside the ϵ-insensitive zone will become support vector. If the gap between the ambiguous data is 2ϵ or more, then each sample becomes a support vector and tries to minimise its approximation error, resulting in the observed fluctuations. Because each

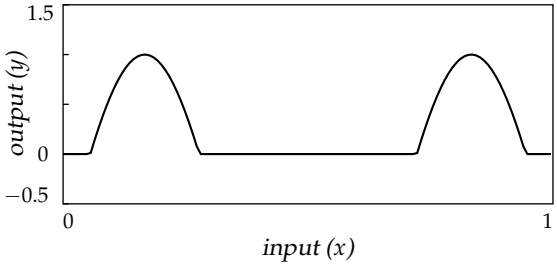


Figure 3.21: One-dimensional test function

support vector tries to minimise its approximation error, and there are as much training samples as there are key samples, similar fluctuations are found for LSSVM(+). The result of KSM for the minimal MSE on the validation set is given in figure 3.20 too. This method is not sensitive to ambiguous data, because KSM includes indicators that minimise the MSE on the complete data set and stop including more indicators if the inclusion is not significant anymore. Furthermore, the limited set of key samples is trained by all the training samples, therewith averaging the ambiguities, as if they are noise.

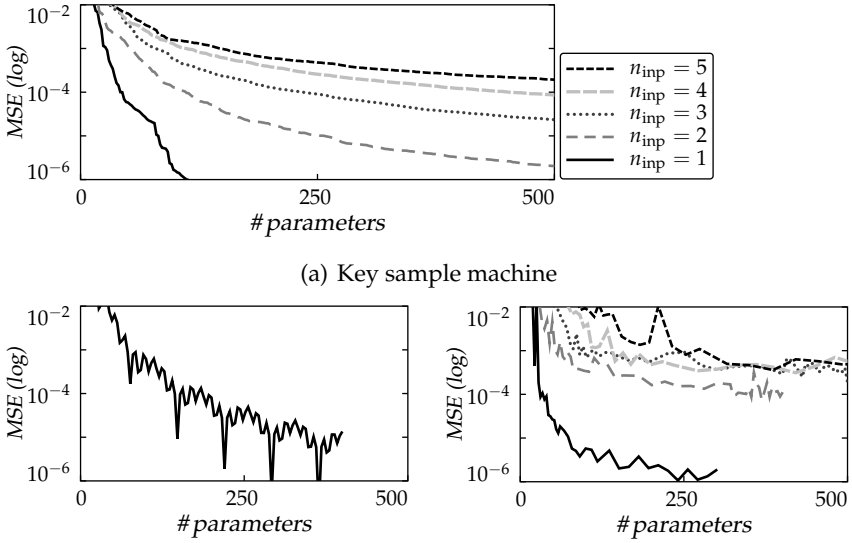
3.5.2 High input dimension

The comparison to test how the different methods deal with several inputs only compares the KSM with the MLP and the BSN. That the other support vector based methods are not considered in this comparison is because the set of functions for these methods is identical to KSM and in the previous results it was shown that the KSM gave the smallest MSE for a given number of indicators.

In this comparison the data is generated by a simple function so that mainly the ability to handle more inputs is evaluated, and not so much the ability to find a complex structure. For the same reason, no noise is included in this approximation. The data is generated by the following function:

$$y(\mathbf{x}) = \frac{1}{n_{\text{inp}}} \sum_{i=1}^{n_{\text{inp}}} 2 \max(\sin(3\pi \mathbf{x}_i) - \tfrac{1}{2}, 0). \tag{3.55}$$

In this equation the subscript of $\mathbf{x}$ denotes the input number. The fraction before the summation keeps the targets within the interval zero and one. This function is plotted in 3.21 for one dimension.



(a) Key sample machine

(b) B-spline network

(c) Artificial neural network

Figure 3.22: MSE for different input dimensions

Although this function is an additive combination of one-dimensional functions, the approximators do not have this information and approximate the function as one n_{inp} dimensional function. So, although the data can be approximated by an approximator that can only approximate relations of the form:

$$\hat{y} = g_1(x_1) + g_2(x_2) + \dots \tag{3.56}$$

for this approximation the methods use the following form:

$$\hat{y} = g(x_1, x_2, \dots). \tag{3.57}$$

This is done because it is often unknown for realistic data sets whether the underlying relation is additive. If the approximator does not have access to this information, it should still be able to approximate it with a reasonable number of structural elements.

The results for the different methods are depicted in figure 3.22. In (a) the MSE of KSM is given, in (b) the MSE for the BSN is depicted. Only one curve can be seen in in (b), because for more-dimensional input spaces, the MSE is larger than 0.3. This is as expected because it was known from previous work that the BSN suffers from the curse of dimensionality. The curves in (c) give the MSE of MLP. Although these experiments

are averaged over five runs, the result is still noisy. The approximations found got often stuck in local minima.

In these figures the MSE is plotted as a function of the number of parameters. The number of parameters is used instead of the number of basis elements to come to a more fair comparison. For the BSN the centres and the weights are considered parameters, for the MLP the weights and for the KSM the inputs of the key samples and the Lagrangian multipliers.

Based on these figures it is clear that the KSM gives a smaller MSE for nearly all numbers of parameters for the different input dimensions. The MSE of KSM and MLP are nearly equal if only a limited number of parameters is used. However, if more parameters are included, the MSE of the KSM decreases more, while the MSE of the MLP remains of the order of 10^{-6}. If more parameters are included, the MLP is more likely to get stuck in a local minimum.

The KSM handles the increasing input dimension well. For a specific MSE, the number of key samples increases with increasing dimension, but not exponentially.

In this comparison concerning high dimensional inputs, the RBF approximators are left out. This is done because an RBF only has an influence in a certain region of the space which makes it prone to the curse of dimensionality. This might be circumvented by altering the support of the basis functions, but in initial approximations this approach nearly always got stuck in local minima. Therefore, it was decided not to go on investigating this approach.

3.5.3 Some notes on computation time

To get an idea of the computation time for an approximation, 2 000 training samples from data set 1, corrupted with additive Gaussian noise with a variance of $(0.2)^2$, were supplied to the different approximators. Approximations were made with different number of indicators, while the computation time and the MSE on the validation set were recorded.

Code is used for the different approximators that has been written by different people, for different purposes. As a result, the degree of computational efficiency differs. Therefore, the results obtained in this comparison *should not be used to draw conclusions*, and are given for indicatory reasons only.

The calculations for SVM were performed with LIBSVM (Chang and Lin, 2003). This program implements an improved version of the SMO algorithm (Keerthi, Shevade, Bhattacharyya, and Murthy, 2001). The

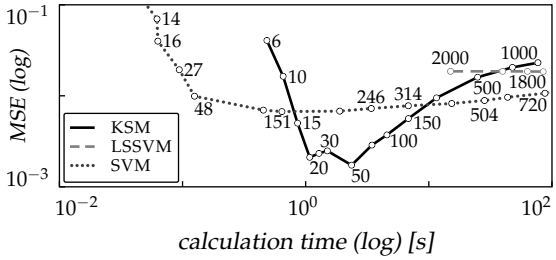


Figure 3.23: Approximation error as a function of the calculation time needed to form the approximation. Numbers in this figure give the number of indicators

calculations for KSM and LSSVM(+) were done with an self-written implementation. This code was written for testing the ideas rather than speed, and is therefore not optimised in terms of computational efficiency .

In figure 3.23 the MSE is plotted as a function of the time needed to find this approximation. The numbers within this figure indicate the number of indicators for the resulting approximation. Note that the y- and the x-axis are both logarithmic.

As of the large differences in the code, no conclusions can be drawn based on this figure. However, some observations could not be refrained from. The calculation time for the KSM grows approximately linear with the number of key samples, while the calculation time of the SVM grows above-linear. For larger numbers of indicators (# indicators > 500), the current implementation of KSM became faster than the calculations for SVM with LIBSVM.

The LSSVM starts with all the training samples as indicators and then starts decreasing this number. Therefore, first the parameter vector needs to be calculated with 2 000 elements. In order to arrive at this solution, a Cholesky decomposition had to be constructed in our algorithm. For a $2\,000 \times 2\,000$ matrix, this takes considerable time, and therefore the first approximation is only found after approximately 15 [s]. After this, the pruning starts. Although the KSM uses the same decomposition for its calculations, it starts with a 1×1 matrix, which grows by the inclusion of each key sample. As a result, the calculation time for few indicators is smaller for KSM than the calculation time for LSSVM.

An advantage of the growing and shrinking methods, is that they

come to their final approximation by increasing or decreasing the number of indicators of the approximation. These intermediate approximations can be stored by which a whole set of models of different complexity is obtained *in one run.* After this run, the user can select one of these models appropriate for the task on hand, based on e.g. a validation set. This makes the construction of a set of models by LSSVM and KSM much faster than for MLP or SVM.

3.5.4 Ease of use

Although the criterion if something is deemed 'easy to use' depends on personal preferences, some comments on this subject will be given.

The KSM makes use of the noise variance in the stopping criterion. The noise variance, even if it is unknown, is a clear and intuitive measure to determine when to stop. The stopping criterion itself was found valuable because it adapts the accuracy of the approximation to the *quality and the quantity* of the data.

In the experiments no use was made of the possibility to include a priori indicators. This possibility is a valuable addition, because a relation known beforehand does not have to be estimated by the potential indicators, thus decreasing the total number of indicators. The variance on the estimate will consequently be smaller.

The possibility to select a kernel function makes all these methods flexible in the relations they can approximate. The disadvantage of this flexibility is that one has to select an appropriate kernel for the problem on hand. If an inappropriate kernel function is selected, a large set of data is required for the prediction. In chapter 5 practical considerations concerning the selection of a kernel will be dealt with.

3.6 Review

This chapter started by treating several off-line function approximators described in literature. From these approximators several ideas have been combined to form a function approximator that better suits our needs. The result is called the Key Sample Machine (KSM).

KSM uses limited samples of the training set to represent the full training set. This idea is also used in the Support Vector Machine (SVM). Because of this *sample-based* approach, the input space is not necessarily divided into regions, as done by e.g. B-splines and Radial Base functions (RBFs), and KSM is therefore less prone to the curse of dimensionality. The samples that are used as summary of the data are called *key samples*.

The interpolation between the key samples is done by a dual indicator function, which is equal to the kernel function of SVM. Because of the use of a kernel function, this method is not limited to only one type of functions by which it can approximate the data, as is the Radial Base Function Network (RBFN) and the Multilayer Perceptron (MLP).

In opposition to SVM, a quadratic cost function is used. This does not inherently result in a sparse solution. In the Least Squares Support Vector Machine (LSSVM), which also uses a quadratic cost function, a sparse subset of training samples is found for prediction, by successively removing those samples from the training set that have little influence on the prediction. This pruning scheme omits valuable information due to these removals. KSM uses a *subset selection scheme* to find the key samples that summarise the data set. However, all training data is used to train the parameters corresponding to the key samples. Because the subset selection is now an explicit step, a selection scheme that is appropriate for the problem on hand can be used. In this thesis the forward selection scheme is used. This scheme includes one key sample a time until a good approximation is found. The calculation time remains small, if the number of key samples remains small. A stopping criterion has been introduced that tests whether the inclusion of an extra key sample is *statistically relevant*. Only if it is, the key sample is included to predict for new inputs. Because of the statistical test, the fitting of noise is improbable.

The result of the above used ideas is that a function approximator is obtained that uses a subset of the presented data to approximate all the data, in which the accuracy of the prediction depends on the *quantity* and the *quality* of the data. If there is more, or better, information, a better approximation is obtained.

The subset selection scheme performed well and in the evaluation it was shown that a suitable structure was found by which the data could be approximated. The method did not fit the noise realisation, unless the noise level was underestimated.

The use of the KSM is intuitive. There is only one design parameter which is the noise estimate. This parameter trades a larger number of key samples and an accurate prediction, with a limited number of key samples and a rough prediction.

The implementation of this method is uncomplicated because many algorithms known from linear algebra support the implementation. For the expansion of the KSM with an extra key sample, the calculations are done recursively and are not built up from scratch. These calculation are treated in appendix B.1.

The comparison with the other support vector based approximators

showed that, for the examples used, the KSM gave a better prediction with fewer key samples. This was especially true if the data was corrupted by noise. Because the KSM used fewer key samples, the computational load to obtain an approximation was less.

3.6.1 Concerning the problem definition

The function approximator has been constructed for the use of control. The KSM should therefore comply with the conditions imposed on an approximator for control.

Real-time constraints

The accuracy of the KSM increases by adding more key samples. If the limitations on memory space or calculation time are reached before the stopping criterion indicates that it is fitting noise, the addition of more key samples can be stopped. This would result in a less accurate approximation than would be possible, but it still complies with the real-time constraints. The prescribed real-time constraints are therefore always complied with. However, it was shown in the evaluation and the comparison that the number of key samples remained limited anyhow. The number of inputs influenced the approximation of KSM, but the number of key samples grows slowly compared with MLP and BSN for a given accuracy.

Generalisation ability

The generalisation ability means that new inputs can be predicted well. Based on the validation sets of the comparisons, it can be concluded that the KSM generalises well as long as the noise estimate is not too small. Even if the data offered is noisy, the KSM can extract data such that a good prediction can be made for unencountered samples.

Comparison with other function approximators has shown that the KSM is able to work with several inputs without any problem.

Based on these observations, the KSM is expected to work well as part of a learning control scheme. In chapter 5, experiments will be conducted to see if KSM meets these expectations.

Four

On-line function approximation

THE FUNCTION APPROXIMATOR in the previous chapter was used in an off-line setting; all the data, obtained in a separate training phase, was presented in a batch and the off-line function approximator approximated this data. The result of this approximation can be used in a control scheme.

In this chapter, a function approximator is developed that can be directly incorporated into a control scheme. From the moment the control scheme starts its operation, this function approximator is supplied with data. The approximator has to continuously enhance its approximation of the relation underlying this data. Because the function approximator is included in the control scheme and has to adapt its approximation continuously, it has to be able to cope with a constant data stream. Approximately each millisecond, for application in which we are interested, a new training sample is produced and should be processed. The data stream is in principle endless, and hence it is not possible to memorise all the samples.

KSM, as introduced in the previous chapter, cannot be used in this on-line setting, because it requires all the data for the selection and testing of a new indicator. However, the idea of summarising the data with a limited subset gave good results and we would like to apply this idea in an on-line approximation as well. The difficulty of implementing this idea in the on-line setting, is how to find the set of key samples *out of a stream of data*, without storing all the previous samples.

In order to come to a solution, the Generalised Least Squares (GLS) and the Recursive Least Squares (RLS) methods are treated. Their theory gives directions on how the key samples can be found and updated

with a constant stream of data. The resulting recursive version of KSM is treated in section 4.2. In section 4.3, this method is evaluated and in section 4.4, it is compared with another on-line function approximator. At the end of the chapter a review is given.

4.1 Background

4.1.1 Generalised Least Squares

Ordinary least squares as treated in the previous chapter, weights all the samples equally for the final approximation. However, if the variance of the additive noise is not the same for all the samples, or if the noise is correlated between the samples, i.e. $E(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^{\mathrm{T}}) \neq \sigma^2\mathbf{I}$, then the use of the normal equations as stated in (3.7) will not give a minimal variance solution, see e.g. (Björck, 1996). The generalised least squares method is introduced to find a minimal variance solution when (3.4) does not hold. This method assigns more weight to samples of which the variance of the additive noise is smaller. This weighting shall be used in the recursive version of KSM.

By use of a linear mapping, the variance on all the samples can be made equal, so that the normal equations can be used to obtain a minimal variance solution (Aitken, 1934). The relation between the targets and the indicators is still given as:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \boldsymbol{\varepsilon} \quad \text{with} \quad E(\boldsymbol{\varepsilon}) = 0. \tag{4.1}$$

Contrary to (3.4), the variance is given as:

$$E(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^{\mathrm{T}}) = \sigma^2\mathbf{V}. \tag{4.2}$$

The diagonal elements of this matrix denote the variance of the noise acting on the samples, while the off-diagonal elements indicate the covariance between the noise samples. The covariance matrix of the noise is equal to the covariance matrix of the samples, due to the additive noise assumption:

$$\begin{aligned} E\Big((\mathbf{y} - E(\mathbf{y}))(\mathbf{y} - E(\mathbf{y}))^{\mathrm{T}}\Big) &= E\Big((\mathbf{X}\mathbf{b} + \boldsymbol{\varepsilon} - \mathbf{X}\mathbf{b})(\mathbf{X}\mathbf{b} + \boldsymbol{\varepsilon} - \mathbf{X}\mathbf{b})^{\mathrm{T}}\Big) \\ &= E(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^{\mathrm{T}}). \end{aligned} \tag{4.3b}$$

So, off-diagonal elements in the variance matrix of the additive noise also indicate that the corresponding targets are correlated.

Because $\mathbf{V}$ is symmetric positive definite (Björck, 1996), there exists a Cholesky decomposition $\mathbf{P}\mathbf{P}^{\mathrm{T}}$ with $\mathbf{P}$ lower triangular so that (Stewart, 1998):

$$\mathbf{P}\mathbf{P}^{\mathrm{T}} = \mathbf{V}. \tag{4.4}$$

Pre-multiplying (4.1) with $\mathbf{P}^{-1}$ results in:

$$\mathbf{P}^{-1}\mathbf{y} = \mathbf{P}^{-1}\mathbf{X}\mathbf{b} + \mathbf{P}^{-1}\boldsymbol{\varepsilon}. \tag{4.5}$$

In this transformed representation the additive noise is given as:

$$\boldsymbol{\psi} = \mathbf{P}^{-1}\boldsymbol{\varepsilon} \quad \text{with} \quad E(\boldsymbol{\psi}) = 0. \tag{4.6}$$

The variance of the additive noise $\boldsymbol{\psi}$ is calculated as:

$$\begin{aligned} E(\boldsymbol{\psi}\boldsymbol{\psi}^{\mathrm{T}}) &= E(\mathbf{P}^{-1}\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^{\mathrm{T}}\mathbf{P}^{-\mathrm{T}}) \\ &= \mathbf{P}^{-1}E(\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^{\mathrm{T}})\mathbf{P}^{-\mathrm{T}} \\ &= \sigma^2\mathbf{P}^{-1}\mathbf{P}\mathbf{P}^{\mathrm{T}}\mathbf{P}^{-\mathrm{T}} \\ &= \sigma^2\mathbf{I}. \end{aligned} \tag{4.7}$$

So, by pre-multiplying $\mathbf{X}\mathbf{b} = \mathbf{y}$ by $\mathbf{P}^{-1}$, the additive noise becomes uncorrelated and equal for all samples. The minimal variance solution of this transformed problem can be obtained by the ordinary least squares method. The minimisation problem is formulated as:

$$\min_{\mathbf{b}} \left\| \mathbf{P}^{-1}(\mathbf{X}\mathbf{b} - \mathbf{y}) \right\|_2^2, \tag{4.8}$$

with the solution:

$$\mathbf{X}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{X}\mathbf{b} = \mathbf{X}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{y}. \tag{4.9}$$

This is obtained by equating the derivatives of the minimisation with respect to $\mathbf{b}$, with zero.

The variance matrix $\mathbf{V}$ allows us to indicate which sample gives more certain information. The inverse of this matrix is called the weight matrix.

4.1.2 Recursive Least Squares

In order to make the approximator recursive, the concepts of RLS are used. The RLS method updates the parameter vector by including a new sample. The new sample is $(\mathbf{x}, y)$. This input induces an indicator

vector $\mathbf{f}$ as described in the section 3.1.1. The normal equations (3.7) hold before the inclusion:

$$\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)\mathbf{b} = \mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{4.10}$$

Including the information of the new sample into the matrices of (4.10) yields:

$$\begin{bmatrix}\mathbf{X}\\ \mathbf{f}\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\mathbf{X}\\ \mathbf{f}\end{bmatrix}(\mathbf{b}+\boldsymbol{\Delta}\mathbf{b}) = \begin{bmatrix}\mathbf{X}\\ \mathbf{f}\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\mathbf{y}\\ y\end{bmatrix}, \tag{4.11}$$

in which $\boldsymbol{\Delta}\mathbf{b}$ represents the modification to the parameter vector due to the inclusion of the new sample. Performing the matrix multiplications gives:

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\Delta}\mathbf{b} + \mathbf{f}^{\mathrm{T}}\mathbf{f}\boldsymbol{\Delta}\mathbf{b} + \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{b} + \mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{b} = \mathbf{X}^{\mathrm{T}}\mathbf{y} + \mathbf{f}^{\mathrm{T}}y. \tag{4.12}$$

Substitution of (4.10) into (4.12):

$$\left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \mathbf{f}^{\mathrm{T}}\mathbf{f}\right)\boldsymbol{\Delta}\mathbf{b} = \mathbf{f}^{\mathrm{T}}(y - \mathbf{f}\mathbf{b}). \tag{4.13}$$

For the calculation of $\boldsymbol{\Delta}\mathbf{b}$ the inverse of $\left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \mathbf{f}^{\mathrm{T}}\mathbf{f}\right)$ should be known. This inverse can be calculated, based on the inverse of $\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)$ which is known from before the inclusion of a new sample, see (4.10). With the use of Woodbury's formula (Golub and Van Loan, 1996):

$$\left(\mathbf{A} + \mathbf{f}^{\mathrm{T}}\mathbf{f}\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{A}^{-1}}{1 + \mathbf{f}\mathbf{A}^{-1}\mathbf{f}^{\mathrm{T}}} \tag{4.14}$$

with

$$\mathbf{A} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right). \tag{4.15}$$

this update can be done without the calculation of a complete inverse. The term $y - \mathbf{f}\mathbf{b}$ is the approximation error that is made by the current parameter vector for the new sample. This scheme is known as RLS (Björck, 1996; Haykin, 1994; Ljung, 1999) and is closely related to Kalman filtering (Ljung, 1999). A similar line of reasoning that is used for RLS, is used to do the update calculations for the recursive KSM.

4.2 Recursive Key Samples Machine

In this section we propose a new on-line function approximator. This method is the recursive version of KSM. In order to come to a recursive version of KSM, two concepts are used:

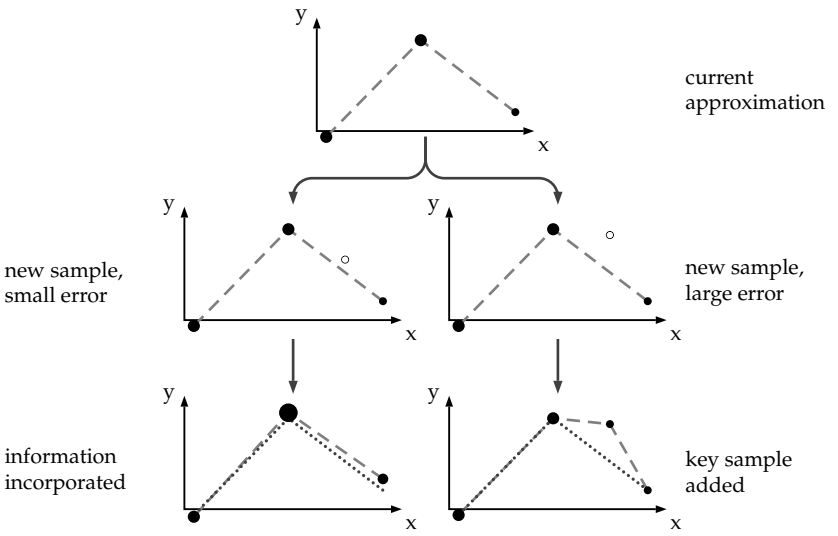


Figure 4.1: Inclusion of a new sample. The new sample can be represented by the present key samples (left path) or it can be used to extend the data structure (right path)

- Assign a weight to each key sample to indicate the multitude of data it represents.
- Make a newly supplied sample a key sample, if the current approximation cannot represent it well enough. Otherwise, use it to fine tune the current key samples.

These ideas are first illustrated by an example.

Example 4.1 (Inclusion of a new sample)
Refer to figure 4.1. Assume that there is already an approximation of previous samples. This approximation is like the one introduced in the previous chapter and given at the top of figure 4.1. The key samples represent a multitude of samples, and a corresponding weight is assigned to the key sample. The key samples are depicted as black dots and the diameter of these dots gives their weight. The approximation is given by the dashed line.

A new sample becomes available and should be incorporated into this approximation. The new sample is indicated by the open dot. Based on some criterion, the sample can become a key sample itself and is used in future predictions. Or, if the sample does not become a key sample, it should be used to fine tune the other key samples. In the left path, the sample is predicted rather

well by the current approximation and the approximation error is therefore assumed to be caused by noise. Therefore, the sample is *not* included as a key sample but it is used to fine tune the present key samples. The approximation after the inclusion of this sample is shown at the bottom-left of figure 4.1. The calculations for the update incorporate the weights. The weight after the update of the three key samples is altered as well as their output, due to the inclusion of the new sample. The previous approximation is indicated by the dotted line, the updated approximation by the dashed line.

In the path on the right, the new sample cannot be predicted correctly by the current approximation. If the approximation error is too large, with respect to the noise level, the sample becomes a key sample. By this inclusion the structure alters, but the weights do not. This key sample is henceforward required for future predictions. ▩

To implement the scheme shown in example 4.1, the following subjects need to be treated:

- How can a weight be assigned to a set of key samples so that they represent the full data set and so that the inclusion of a new sample alters the approximation correctly.
- How can we update the set of key samples to incorporate a new sample.
- When should a sample become a key sample.

In this section, these subjects are treated. The resulting algorithm is given on page 100 as algorithm 4.2 .

4.2.1 Weighting of the key samples

A reduced set of key samples should represent a full set of data by making use of weighting the key samples. For the calculation of these weights a distinction is made between two sets:

Reduced set: This set contains only the k key samples.

Full set: This set contains all the training data.

As in the previous chapter, each sample induces a dual indicator function. Only the dual indicator functions of the key samples are used in the actual approximation. Because the reduced set contains only these k

key samples, the indicator matrix of the reduced set is given as:

$$\mathbf{X}_{\text{ks}} = \begin{bmatrix} \tilde{f}_1(\mathbf{x}_{\text{ks},1}) & \tilde{f}_2(\mathbf{x}_{\text{ks},1}) & \dots & \tilde{f}_k(\mathbf{x}_{\text{ks},1}) \\ \tilde{f}_1(\mathbf{x}_{\text{ks},2}) & \tilde{f}_2(\mathbf{x}_{\text{ks},2}) & \dots & \tilde{f}_k(\mathbf{x}_{\text{ks},2}) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_{\text{ks},k}) & \tilde{f}_2(\mathbf{x}_{\text{ks},k}) & \dots & \tilde{f}_k(\mathbf{x}_{\text{ks},k}) \end{bmatrix}. \tag{4.16}$$

Note that $\tilde{f}_i(\mathbf{x}_j) = \tilde{f}_j(\mathbf{x}_i)$ because it represents an innerproduct of key sample i and j in some feature space.

The indicator functions of the full set are equal to the indicator functions of the reduced set, i.e. the same set of functions can be realised. However, N training samples are included in the indicator matrix of the full data set:

$$\mathbf{X} = \begin{bmatrix} \tilde{f}_1(\mathbf{x}_1) & \tilde{f}_2(\mathbf{x}_1) & \dots & \tilde{f}_k(\mathbf{x}_1) \\ \tilde{f}_1(\mathbf{x}_2) & \tilde{f}_2(\mathbf{x}_2) & \dots & \tilde{f}_k(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_k) & \tilde{f}_2(\mathbf{x}_k) & \dots & \tilde{f}_k(\mathbf{x}_k) \\ \tilde{f}_1(\mathbf{x}_{k+1}) & \tilde{f}_2(\mathbf{x}_{k+1}) & \dots & \tilde{f}_k(\mathbf{x}_{k+1}) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_N) & \tilde{f}_2(\mathbf{x}_N) & \dots & \tilde{f}_k(\mathbf{x}_N) \end{bmatrix}. \tag{4.17}$$

In order to let the key samples represent the complete data set, a weight is introduced so that the normal equations and the weighted normal equations become identical. The weighted (4.9) and unweighted (3.7) normal equations are repeated below for convenience sake:

$$\mathbf{X}^{\text{T}}\mathbf{X}\mathbf{b} = \mathbf{X}^{\text{T}}\mathbf{y}, \tag{4.18a}$$

$$\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}}\mathbf{b} = \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{y}_{\text{ks}}. \tag{4.18b}$$

$\mathbf{V}^{-1}$ is the covariance matrix which is used in RLS for the weighting of the samples. When the left-hand sides of the equations in (4.18) are made identical, it follows that:

$$\mathbf{X}^{\text{T}}\mathbf{X}\mathbf{b} = \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}}\mathbf{b} \;\Rightarrow\; \mathbf{V}^{-1} = \mathbf{X}_{\text{ks}}^{-\text{T}}\mathbf{X}^{\text{T}}\mathbf{X}\mathbf{X}_{\text{ks}}^{-1}, \tag{4.19}$$

which gives a matrix that can be used for weighting the key samples. The same for the right-hand side:

$$\mathbf{X}^{\text{T}}\mathbf{y} = \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{y}_{\text{ks}}, \tag{4.20a}$$

$$\mathbf{y}_{\text{ks}} = \mathbf{V}\mathbf{X}_{\text{ks}}^{-\text{T}}\mathbf{X}^{\text{T}}\mathbf{y} \tag{4.20b}$$

$$= \mathbf{X}_{\text{ks}}(\mathbf{X}^{\text{T}}\mathbf{X})^{-1}\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{X}_{\text{ks}}^{-\text{T}}\mathbf{X}^{\text{T}}\mathbf{y} \tag{4.20c}$$

$$= \mathbf{X}_{\text{ks}}(\mathbf{X}^{\text{T}}\mathbf{X})^{-1}\mathbf{X}^{\text{T}}\mathbf{y}. \tag{4.20d}$$

This makes the outputs of the key samples $\mathbf{y}_{\text{ks}}$ equal to the prediction of the full data estimator at the key samples.

It is interesting to see that by setting the weighted and unweighted normal equations identical, the variance of the prediction for the weighted and unweighed approximation, as well as the predictions, are identical. The prediction of the full approximator is given in (4.20d). The prediction of the reduced approximator is given as:

$$\hat{\mathbf{y}}_{\text{ks}}(\mathbf{x}_{\text{ks}}) = \mathbf{X}_{\text{ks}}\mathbf{b}. \tag{4.21}$$

Substituting (4.18b) for $\mathbf{b}$ into this equation yields:

$$\begin{aligned}
\hat{\mathbf{y}}_{\text{ks}}(\mathbf{x}_{\text{ks}}) &= \mathbf{X}_{\text{ks}}(\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}})^{-1}\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\hat{\mathbf{y}}_{\text{ks}} && (4.22a)\\
&= \mathbf{X}_{\text{ks}}(\mathbf{X}^{\text{T}}\mathbf{X})^{-1}\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{X}_{\text{ks}}^{-\text{T}}\mathbf{X}^{\text{T}}\mathbf{X}\mathbf{X}_{\text{ks}}^{-1}\hat{\mathbf{y}}_{\text{ks}} && (4.22b)\\
&= \hat{\mathbf{y}}_{\text{ks}}. && (4.22c)
\end{aligned}$$

This is, as expected, equal to the prediction of the full approximation. The variance of the full approximation for some input is (Draper and Smith, 1998):

$$\mathbf{f}(\mathbf{x})\left(\mathbf{X}^{\text{T}}\mathbf{X}\right)^{-1}\mathbf{f}^{\text{T}}(\mathbf{x})\sigma^2, \tag{4.23}$$

and the same variance is given for the reduced approximation as:

$$\mathbf{f}(\mathbf{x})\left(\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}}\right)^{-1}\mathbf{f}^{\text{T}}(\mathbf{x})\sigma^2, \tag{4.24}$$

which is identical for the calculated value of $\mathbf{V}$ in (4.19). So, identical predictions and variances are found for the full unweighted data and the reduced weighted data.

Example 4.2 (Calculation of weight matrix)
The data set that is used in this example is given in table 4.1(a). The inputs of the key samples are $x_{\text{ks},1} = 0, x_{\text{ks},2} = 1$ and the approximation is done by infinite splines. The dual indicator function for infinite spline is given as:

$$\tilde{f}_i(x) = 1 + \min(x, x_{\text{ks},i}), \tag{4.25}$$

see example 3.3. The indicator matrices follow from (4.16) and (4.17):

Table 4.1: Data for the weight calculation

(a) Data for example 4.2

x	y
0	1
0	2
1	3
1	4

(b) Data for example 4.3

x	y
0	1
0.2	2
1	3
1	4

$$\mathbf{X} = \begin{bmatrix} \tilde{f}_1(x_1) & \tilde{f}_2(x_1) \\ \tilde{f}_1(x_2) & \tilde{f}_2(x_2) \\ \tilde{f}_1(x_3) & \tilde{f}_2(x_3) \\ \tilde{f}_1(x_4) & \tilde{f}_2(x_4) \end{bmatrix} \tag{4.26a}$$

$$= \begin{bmatrix} 1+\min(x_1, x_{\text{ks},1}) & 1+\min(x_1, x_{\text{ks},2}) \\ 1+\min(x_2, x_{\text{ks},1}) & 1+\min(x_2, x_{\text{ks},2}) \\ 1+\min(x_3, x_{\text{ks},1}) & 1+\min(x_3, x_{\text{ks},2}) \\ 1+\min(x_4, x_{\text{ks},1}) & 1+\min(x_4, x_{\text{ks},2}) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}, \tag{4.26b}$$

and

$$\mathbf{X}_{\text{ks}} = \begin{bmatrix} \tilde{f}_1(x_{\text{ks},1}) & \tilde{f}_2(x_{\text{ks},1}) \\ \tilde{f}_1(x_{\text{ks},2}) & \tilde{f}_2(x_{\text{ks},2}) \end{bmatrix} \tag{4.27a}$$

$$= \begin{bmatrix} 1+\min(x_{\text{ks},1}, x_{\text{ks},1}) & 1+\min(x_{\text{ks},1}, x_{\text{ks},2}) \\ 1+\min(x_{\text{ks},2}, x_{\text{ks},1}) & 1+\min(x_{\text{ks},2}, x_{\text{ks},2}) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}. \tag{4.27b}$$

The value for the weight matrix and the key sample outputs follow from (4.19) and (4.20d) as:

$$\mathbf{V}^{-1} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \mathbf{V} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \mathbf{y}_{\text{ks}} = \begin{bmatrix} 1.5 \\ 3.5 \end{bmatrix}. \tag{4.28}$$

This indicates that the two key samples are twice as important as a single sample. Or in terms of the variance matrix, the variance of the key samples is only half that of each single sample. This is quite obvious, because the key samples represent two samples with the same input. The key samples' targets lie between the targets of the samples that they represent. ▩

Example 4.3 (Covariance in weight matrix)
A second example uses the data set of table 4.1(b). The same key samples are used to represent the data. For this data set the following results are found:

$$\mathbf{V}^{-1} = \begin{bmatrix} 1.64 & 0.16 \\ 0.16 & 2.04 \end{bmatrix}, \mathbf{V} = \begin{bmatrix} 0.61 & -0.048 \\ -0.048 & 0.49 \end{bmatrix}, \mathbf{y}_{\text{ks}} = \begin{bmatrix} 1.24 \\ 3.53 \end{bmatrix}. \tag{4.29}$$

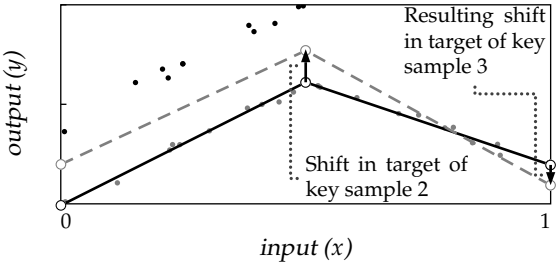


Figure 4.2: Influence of the covariance on a key sample

In the covariance matrix the off-diagonal terms became non-zero. This means that the key samples are correlated, which is caused by the fact that the input of the second sample in the data set is in between the key samples. The result of this covariance term in the matrix $\mathbf{V}^{-1}$, is that if the first key sample target increases due to future inclusions, the target of the second key sample decreases.

This effect is shown in figure 4.2 for a larger data set. There are three key samples indicated by the open circles. First an approximation is made for the gray samples. The approximation is given by the black solid line. The weights of the key samples are correlated. A set of new samples is offered to the approximator, which are indicated by the black dots. All of these samples are located between 0 and 0.5. Due to these samples the left part of the approximation increases. The required calculations for this update will be treated next. As a result of the covariances in the weight matrix, the rightmost key sample will decrease when the middle key sample is increased. The information of old samples is incorporated in the weight matrix. ▩

Summarising: the weight matrix makes the normal equations of the full data set identical to the weighted normal equations. As a result, the prediction and the variance based on the full data set is equal to the prediction and the variance of the weighted reduced data set. The diagonal elements of the weight matrix give the importance of the corresponding key sample. The off-diagonal elements of the weight matrix's inverse indicate the covariance between key samples. Due to this covariance, an increase of the target of one key sample influences the target of the other key samples. This makes it possible to omit training samples, because the information of this data is incorporated in the weight matrix for future updates of the approximation.

4.2.2 Growing and shrinking

This section deals with the calculations necessary for the updates of the approximation. Three cases are considered separately:

Case 1) A new sample is represented by the key samples.

Case 2) A new sample becomes a key sample.

Case 3) A key sample is removed.

The calculations shown here are *not* efficient, nor are they numerically stable, but they do show what happens. For a more efficient and numerically stable implementation, see appendix B.

Fine tuning the key samples

The first case to be considered is when the offered sample does not become a key sample and is therefore represented by the already present key samples. This is illustrated by the left path of figure 4.1. Because the same key samples are used to represent the data before and after the update, only the weight and the key sample targets are altered by the update.

The normal equations before and after the inclusion of the sample should be identical to the weighted normal equations. The situation before the inclusion is given as (4.18a), and is repeated here:

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{X}_{\mathrm{ks}}. \tag{4.30}$$

Because the key samples are kept equal, $\mathbf{X}_{\mathrm{ks}}$ does not change. After the inclusion, (4.30) has to hold for the updated matrices:

$$\bar{\mathbf{X}}^{\mathrm{T}}\bar{\mathbf{X}} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\mathbf{X}_{\mathrm{ks}}. \tag{4.31}$$

A bar over a matrix is used to indicate that it is an updated matrix. Partitioning $\bar{\mathbf{X}}$ as done with the calculations for the RLS results in:

$$\begin{bmatrix}\mathbf{X}^{\mathrm{T}} & \mathbf{f}^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\mathbf{X}\\ \mathbf{f}\end{bmatrix} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\mathbf{X}_{\mathrm{ks}}, \tag{4.32a}$$

$$\mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{X}_{\mathrm{ks}}^{-1} + \mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1} = \bar{\mathbf{V}}^{-1}, \tag{4.32b}$$

$$\mathbf{V}^{-1} + \mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1} = \bar{\mathbf{V}}^{-1}. \tag{4.32c}$$

$\mathbf{f}$ is the indicator vector for the new sample. This update shows that the updated weight matrix can be derived from the previous weight matrix. This update, and all the coming ones are summarised in algorithm 4.1 on page 91.

This is an exact update. There is no difference if all the data is given at once or if it is updated with this scheme, because no information is lacking for the update (4.32).

Next to the weight matrix the key targets have to be updated. Before the update, (4.20a) holds:

$$\mathbf{X}^{\mathrm{T}}\mathbf{y} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{y}_{\mathrm{ks}}. \tag{4.33}$$

This equation also has to hold after updating the matrices:

$$\bar{\mathbf{X}}^{\mathrm{T}}\bar{\mathbf{y}} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\bar{\mathbf{y}}_{\mathrm{ks}}. \tag{4.34}$$

The updated matrices are denoted by a bar, and the partitioning of $\bar{\mathbf{X}}$ is reused. The updated target vector of the full data set is partitioned as done with RLS, using $\mathbf{y}$ for the previous targets and y for the new target:

$$\begin{aligned}
\begin{bmatrix}\mathbf{X}^{\mathrm{T}} & \mathbf{f}^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\mathbf{y}\\ y\end{bmatrix} &= \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\bar{\mathbf{y}}_{\mathrm{ks}}, && (4.35a)\\
\mathbf{X}^{\mathrm{T}}\mathbf{y} + \mathbf{f}^{\mathrm{T}}y &= \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\bar{\mathbf{y}}_{\mathrm{ks}}, && (4.35b)\\
\mathbf{X}^{\mathrm{T}}\mathbf{y} + \mathbf{f}^{\mathrm{T}}y &= \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\left(\mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1} + \mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{X}_{\mathrm{ks}}^{-1}\right)(\mathbf{y}_{\mathrm{ks}} + \Delta\mathbf{y}_{\mathrm{ks}}), && \\
\mathbf{f}^{\mathrm{T}}y &= \mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1}\mathbf{y}_{\mathrm{ks}} + \mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1}\Delta\mathbf{y}_{\mathrm{ks}} + \mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{X}_{\mathrm{ks}}^{-1}\Delta\mathbf{y}_{\mathrm{ks}}, && \\
\mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\left(y - \mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1}\mathbf{y}_{\mathrm{ks}}\right) &= \bar{\mathbf{V}}^{-1}\Delta\mathbf{y}_{\mathrm{ks}}. && (4.35e)
\end{aligned}$$

The update is rendered in algorithm 4.1(a) on page 91. It shows that there is an update for the key sample targets, but it is better to use the implementation given in appendix B.

Example 4.4 (Updating the key samples)
Assume we have the approximation of the data of example 4.2. The weight matrix and the key sample indicator matrix are given as before as:

$$\mathbf{X}_{\mathrm{ks}} = \begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}, \mathbf{V}^{-1} = \begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}, \mathbf{y}_{\mathrm{ks}} = \begin{bmatrix}1.5\\ 3.5\end{bmatrix}. \tag{4.36}$$

The key samples are $x_{\mathrm{ks},1} = 0$ and $x_{\mathrm{ks},2} = 1$. The new sample presented to the approximator is $(x, y) = (1, 5)$. With this new sample, the weight matrix is updated as:

$$\begin{aligned}
\bar{\mathbf{V}}^{-1} &= \mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1} + \mathbf{V}^{-1} && (4.37a)\\
&= \begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}^{-\mathrm{T}}\begin{bmatrix}1 & 2\end{bmatrix}\begin{bmatrix}1 & 2\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}1 & 1\\ 1 & 2\end{bmatrix}^{-1} + \begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix} && (4.37b)\\
&= \begin{bmatrix}2 & 0\\ 0 & 3\end{bmatrix}. && (4.37c)
\end{aligned}$$

which shows that the second key sample represents three training samples after the update. The update of the targets for the key samples yield:

$$\begin{aligned}
\mathbf{X}_{\text{ks}}^{-\text{T}}\mathbf{f}^{\text{T}}(y - \mathbf{f}\mathbf{X}_{\text{ks}}^{-1}\mathbf{y}_{\text{ks}}) &= \bar{\mathbf{V}}^{-1}\Delta\mathbf{y}_{\text{ks}}, && (4.38a)\\
\begin{bmatrix} 0 \\ 1.5 \end{bmatrix} &= \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}\Delta\mathbf{y}_{\text{ks}}, && (4.38b)\\
\Delta\mathbf{y}_{\text{ks}} &= \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}. && (4.38c)
\end{aligned}$$

So, two key samples $(x_{\text{ks},1}, y_{\text{ks},1}) = (0, 1.5)$ and $(x_{\text{ks},2}, y_{\text{ks},2}) = (1, 4)$ with the weight given in (4.37) represent the data. ▩

Adding a new key sample

The second situation that can occur is that the new sample cannot be represented by the current key samples and should become a key sample itself. This is shown in the path on the right of figure 4.1. In this case the number of key samples increases. Therefore, the key sample indicator matrix is altered. Again, the weight equation (4.18a) before the update holds:

$$\mathbf{X}^{\text{T}}\mathbf{X} = \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}}. \tag{4.39}$$

Because the number of key samples increases, the matrices grow. Introducing the updated matrices in partitioned form:

$$\bar{\mathbf{X}} = \begin{bmatrix} \mathbf{X} & \mathbf{z} \\ \mathbf{f} & \phi \end{bmatrix}, \bar{\mathbf{X}}_{\text{ks}} = \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} \\ \mathbf{f} & \phi \end{bmatrix}, \bar{\mathbf{V}}^{-1} = \begin{bmatrix} \mathbf{V}^{-1} & \mathbf{v}^{\text{T}} \\ \mathbf{v} & \nu^{-1} \end{bmatrix}, \tag{4.40}$$

in which $\phi = \tilde{f}_{k+1}(x_{\text{ks},k+1}) = k(x_{\text{ks},k+1}, x_{\text{ks},k+1})$ is the value of the new dual indicator function for the new key sample. $\mathbf{z}$ is a vector containing the values of all the previous training samples with the new indicator function *and this vector is unknown*. After the inclusion of the key sample the relation equating the variances have to hold for the updated matrices:

$$\begin{bmatrix} \mathbf{X} & \mathbf{z} \\ \mathbf{f} & \phi \end{bmatrix}^{\text{T}} \begin{bmatrix} \mathbf{X} & \mathbf{z} \\ \mathbf{f} & \phi \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} \\ \mathbf{f} & \phi \end{bmatrix}^{\text{T}} \begin{bmatrix} \mathbf{V}^{-1} & \mathbf{v}^{\text{T}} \\ \mathbf{v} & \nu^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} \\ \mathbf{f} & \phi \end{bmatrix}. \tag{4.41}$$

As $\mathbf{z}$ is unknown, the value of $\mathbf{v}$ cannot be calculated. However, some value of $\mathbf{v}$ should be decided on. The assumption used henceforward is that $\mathbf{v} = 0$. Due to this assumption, the update is *not exact.*

Assumption 4 (Uncorrelated new key sample) *The newly included key sample is uncorrelated to the current key samples:* $\mathbf{v} = 0$.

In appendix C the implications of this assumption are treated. Without loss of generality, we set $\nu^{-1} = 1$.

If the key samples represent numerous samples, the diagonal elements of $\mathbf{V}^{-1}$ are much larger than 1. This implies that the output of the newly included key sample alters sooner than the output of the other key samples. The necessary steps for this update are indicated in algorithm 4.1(b). The target of the new key sample can be set freely; throughout this thesis, the target of the newly added key sample coincides with the target of the presented sample.

Omitting a key sample

Apart from the inclusion of a new sample into the approximator, as a key sample or not, there is also the possibility to *omit* an existing key sample from the set of key samples. This can be useful because due to the inclusion of new key samples, old key samples can become superfluous. When we omit these, the matrices will be smaller, resulting in less computation time and memory requirements. Without loss of generality, we assume that key sample k is omitted.

Before the omission of the key sample (4.18a) again has to hold:

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{X}_{\mathrm{ks}}. \tag{4.42}$$

After the update this equation has to hold for the updated matrices. The matrices before the removal are partitioned as follows:

$$\mathbf{X} = \begin{bmatrix}\bar{\mathbf{X}} & \mathbf{z}\end{bmatrix}, \mathbf{X}_{\mathrm{ks}} = \begin{bmatrix}\bar{\mathbf{X}}_{\mathrm{ks}} & \mathbf{f}^{\mathrm{T}} \\ \mathbf{f} & \phi\end{bmatrix}, \mathbf{V}^{-1} = \begin{bmatrix}\mathbf{W}^{-1} & \mathbf{v} \\ \mathbf{v}^{\mathrm{T}} & \nu^{-1}\end{bmatrix}. \tag{4.43}$$

Note that the partitioning of $\mathbf{X}$ and $\mathbf{X}_{\mathrm{ks}}$ already contains the matrices that will result after the removal of the key sample, i.e. $\mathbf{X}$ and $\mathbf{X}_{\mathrm{ks}}$ make use of $\bar{\mathbf{X}}$ and $\bar{\mathbf{X}}_{\mathrm{ks}}$. This can be done, because the elements of the indicator matrices do not alter because of to the removal. However, in the partitioning of $\mathbf{V}^{-1}$, the matrix $\bar{\mathbf{V}}^{-1}$ cannot be found. This is because the weights of the remaining key samples *do* alter due to the removal, as the calculations will show.

Substituting the partitionings of (4.43) in (4.42) and investigating the upper left block after the multiplications gives:

$$\bar{\mathbf{X}}^{\mathrm{T}}\bar{\mathbf{X}} = \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{W}^{-1}\mathbf{X}_{\mathrm{ks}} + \mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\bar{\mathbf{X}}_{\mathrm{ks}} + \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{v}\mathbf{f} + \mathbf{f}^{\mathrm{T}}\nu^{-1}\mathbf{f}. \tag{4.44}$$

This equation holds before the update. After the omission of the key sample the updated version of (4.42) has to hold for the updated matrices:

$$\bar{\mathbf{X}}^{\mathrm{T}}\bar{\mathbf{X}} = \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\bar{\mathbf{X}}_{\mathrm{ks}}. \tag{4.45}$$

Substituting the right-hand side of (4.44) into (4.45) gives:

$$\begin{aligned}
\mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\mathbf{X}_{\mathrm{ks}} &= \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{W}^{-1}\mathbf{X}_{\mathrm{ks}} + \mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{X}_{\mathrm{ks}} + \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{v}\mathbf{f} + \mathbf{f}^{\mathrm{T}}\nu^{-1}\mathbf{f}, \quad (4.46a)\\
\bar{\mathbf{V}}^{-1} &= \mathbf{W}^{-1} + \mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}} + \mathbf{v}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1} + \mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\nu^{-1}\mathbf{f}\mathbf{X}_{\mathrm{ks}}^{-1}.
\end{aligned}$$

Before the update the weight matrix for the remaining key samples was equal to $\mathbf{W}^{-1}$, while after the update the weight matrix is given as $\bar{\mathbf{V}}^{-1}$. In the equation above, it can be seen that these are not equal and that the weight of the remaining key samples alter due to the removal of a key sample.

Next to the update of the weight matrix, the key targets should get an updated value. Before the update, the following holds:

$$\mathbf{X}^{\mathrm{T}}\mathbf{y} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{y}_{\mathrm{ks}}. \tag{4.47}$$

Rewriting this in partitioned form gives:

$$\begin{bmatrix}\bar{\mathbf{X}} & \mathbf{z}\end{bmatrix}^{\mathrm{T}}\mathbf{y} = \begin{bmatrix}\bar{\mathbf{X}}_{\mathrm{ks}} & \mathbf{f}^{\mathrm{T}}\\ \mathbf{f} & \phi\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\mathbf{W}^{-1} & \mathbf{v}\\ \mathbf{v}^{\mathrm{T}} & \nu^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{t}\\ y_{\mathrm{ks}}\end{bmatrix}. \tag{4.48}$$

Investigating the upper block of this matrix equality after multiplication gives:

$$\bar{\mathbf{X}}^{\mathrm{T}}\mathbf{y} = \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{W}^{-1}\mathbf{t} + \mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{t} + \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{v}y_{\mathrm{ks}} + \mathbf{f}^{\mathrm{T}}\nu^{-1}y_{\mathrm{ks}}. \tag{4.49}$$

This equation holds before the omission. After the omission, (4.47) has to hold for the updated matrices:

$$\bar{\mathbf{X}}^{\mathrm{T}}\mathbf{y} = \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\bar{\mathbf{y}}_{\mathrm{ks}}. \tag{4.50}$$

Just as the weight update can be calculated, so can the change in the output of the key samples. Substituting the right-hand side of (4.49) into (4.50):

$$\begin{aligned}
\bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\bar{\mathbf{V}}^{-1}\bar{\mathbf{y}}_{\mathrm{ks}} &= \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{W}^{-1}\mathbf{t} + \mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{t} + \bar{\mathbf{X}}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{v}y_{\mathrm{ks}} + \mathbf{f}^{\mathrm{T}}\nu^{-1}y_{\mathrm{ks}}, \quad (4.51a)\\
\bar{\mathbf{V}}^{-1}\mathbf{t} + \bar{\mathbf{V}}^{-1}\Delta\mathbf{t} &= \mathbf{W}^{-1}\mathbf{t} + \bar{\mathbf{X}}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{t} + \mathbf{v}y_{\mathrm{ks}} + \bar{\mathbf{X}}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\nu^{-1}y_{\mathrm{ks}}, \quad (4.51b)\\
\bar{\mathbf{V}}^{-1}\Delta\mathbf{t} &= \mathbf{v}y_{\mathrm{ks}} + \bar{\mathbf{X}}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\nu^{-1}y_{\mathrm{ks}} - (\bar{\mathbf{V}}^{-1} - \mathbf{W}^{-1} - \bar{\mathbf{X}}_{\mathrm{ks}}^{-\mathrm{T}})\mathbf{t},\\
\bar{\mathbf{V}}^{-1}\Delta\mathbf{t} &= (\mathbf{v} + \bar{\mathbf{X}}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\nu^{-1})y_{\mathrm{ks}} - (\mathbf{v}\mathbf{f}\bar{\mathbf{X}}_{\mathrm{ks}}^{-1} + \bar{\mathbf{X}}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{v}\mathbf{f}\bar{\mathbf{X}}_{\mathrm{ks}}^{-1})\mathbf{t},
\end{aligned}$$

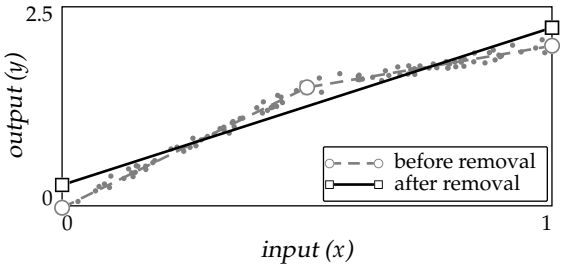


Figure 4.3: Removal of a key sample

with $\bar{\mathbf{y}}_{\text{ks}} = \mathbf{t} + \Delta\mathbf{t}$. The update algorithm described in appendix B gives a faster update. Both updates are given in algorithm 4.1(c).

Example 4.5 (Omitting a key sample)
Let us illustrate the use of this weight update by the omission of a key sample. The training data is plotted in figure 4.3. The underlying relation consists of two straight lines. The data is approximated by key samples at $x_{\text{ks},1} = 0, x_{\text{ks},2} = 1$ and $x_{\text{ks},3} = 0.5$, which are connected by lines. The indicator and weight matrix as well as the key targets are given as:

$$\mathbf{X}_{\text{ks}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1.5 \\ 1 & 1.5 & 1.5 \end{bmatrix}, \mathbf{V}^{-1} = \begin{bmatrix} 13.8 & 0 & 8.8 \\ 0 & 16.9 & 9.3 \\ 8.8 & 9.3 & 33.0 \end{bmatrix}, \mathbf{y}_{\text{ks}} = \begin{bmatrix} -0.01 \\ 2.01 \\ 1.49 \end{bmatrix}. \quad (4.52)$$

In this $\mathbf{X}_{\text{ks}}$ follows from the choice of the key samples, while the values of $\mathbf{V}^{-1}$ and $\mathbf{y}_{\text{ks}}$ are determined by the supplied training data. The resulting approximation is shown by the gray dashed line in figure 4.3. The key samples are shown by open dots.

We decide that the key sample $x_{\text{ks},3} = 0.5$ is omitted from the set of key samples. By the use of algorithm 4.1(c) the new matrices become:

$$\bar{\mathbf{X}}_{\text{ks}} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}, \bar{\mathbf{V}}^{-1} = \begin{bmatrix} 30.9 & 17.3 \\ 17.3 & 34.5 \end{bmatrix}, \bar{\mathbf{y}}_{\text{ks}} = \begin{bmatrix} 0.26 \\ 2.24 \end{bmatrix}. \quad (4.53)$$

The resulting approximation is shown by the black solid line in figure 4.3. The new key samples are indicated by black squares. It can be seen that the approximation after the omission incorporates the information of that key sample into the weight of the other samples, so that no information is lost. This update is exact. ▩

Algorithm 4.1: Update of weight matrix and key sample indicator matrix for inclusion or omission.

Given: $\mathbf{x}_{\text{ks}}, \mathbf{X}_{\text{ks}}, \mathbf{y}_{\text{ks}}, \mathbf{V}^{-1}$ and a new sample $(\mathbf{x}, y)$

1: form $\mathbf{f}$
2: solve $\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{l} = \mathbf{f}^{\text{T}}$ for $\mathbf{l}$
3: $\mathbf{V}^{-1} \leftarrow \mathbf{V}^{-1} + \mathbf{l}^{\text{T}}\mathbf{l}$
4: $\mathbf{y}_{\text{ks}} \leftarrow \mathbf{V}\big(\mathbf{y}_{\text{ks}} + \mathbf{l}(y - \mathbf{l}^{\text{T}}\mathbf{y}_{\text{ks}})\big)$

(a) Inclusion of sample, set of key samples unchanged

Given: $\mathbf{x}_{\text{ks}}, \mathbf{X}_{\text{ks}}, \mathbf{y}_{\text{ks}}, \mathbf{V}^{-1}$ and a new sample $(\mathbf{x}, y)$

1: form $\mathbf{f}$
2: $\phi = f_{k+1}(\mathbf{x})$
3: $\mathbf{X}_{\text{ks}} \leftarrow \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} \\ \mathbf{f} & \phi \end{bmatrix}$
4: $\mathbf{V}^{-1} \leftarrow \begin{bmatrix} \mathbf{V}^{-1} & 0 \\ 0 & 1 \end{bmatrix}$
5: $\mathbf{y}_{\text{ks}} \leftarrow \begin{bmatrix} \mathbf{y}_{\text{ks}} \\ y \end{bmatrix}$

(b) Inclusion of sample, set of key samples extended

Given: $\mathbf{x}_{\text{ks}}, \mathbf{X}_{\text{ks}}, \mathbf{y}_{\text{ks}}, \mathbf{V}^{-1}$

1: form $\mathbf{f}$ for omitted key sample
2: partition $\mathbf{V}^{-1} \rightarrow \begin{bmatrix} \mathbf{W}^{-1} & \mathbf{v} \\ \mathbf{v}^{\text{T}} & \nu^{-1} \end{bmatrix}$
3: partition $\mathbf{X}_{\text{ks}} \rightarrow \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} \\ \mathbf{f} & \phi \end{bmatrix}$
4: partition $\mathbf{y}_{\text{ks}} \rightarrow \begin{bmatrix} \mathbf{y}_{\text{ks}} \\ y \end{bmatrix}$
5: solve $\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{l} = \mathbf{f}^{\text{T}}$ for $\mathbf{l}$
6: $\mathbf{V}^{-1} \leftarrow \mathbf{W}^{-1} + \mathbf{l}\mathbf{v}^{\text{T}} + \mathbf{v}\mathbf{l}^{\text{T}} + \mathbf{l}\nu^{-1}\mathbf{l}^{\text{T}}$
7: $\mathbf{y}_{\text{ks}} \leftarrow \mathbf{V}\big(\mathbf{v}y + \mathbf{l}\nu^{-1}y - (\mathbf{v}\mathbf{l}^{\text{T}} + \mathbf{l}\nu^{-1}\mathbf{l}^{\text{T}})\mathbf{y}_{\text{ks}}\big)$

(c) Omission of a key sample, set of key samples reduced

4.2.3 Selection criteria

Selection of a key sample

Whenever a new sample is inputted to the learning mechanism, it should be decided whether this sample can be approximated by the current set of key samples, or if it should become a key sample itself. To test this, we test the same hypothesis as for the off-line approximation: '$\alpha_i = 0$.' Again, if this hypothesis is rejected, the corresponding sample becomes a key sample.

If the new parameter is equal to zero, then is the difference of the summed squared error before and after inclusion is χ_1^2 distributed:

$$\frac{S^2}{\sigma^2} \sim \chi_1^2 \quad \Leftrightarrow \quad \alpha_i = 0. \tag{4.54}$$

Remark that the error reduction cannot be tested on all the samples because old samples are not stored. However, it can be calculated what the *weighted* change is at the key samples *plus* the residual of the newly supplied sample. Because the key samples are used as representatives, this weighted error change is used to represent the change in error of the processed training samples. The probability that a certain realisation of the error reduction is found, if the new parameter is zero, is calculated as follows:

$$P\left(\frac{S^2}{\sigma^2} \geq z_\zeta \middle| \alpha_i = 0\right) < \zeta. \tag{4.55}$$

In (4.55) ζ denotes the probability that a realisation of z_ζ or larger is found. ζ is called the *significance level* and the corresponding value of z_ζ follows from the χ_1^2 distribution. If the probability for the found error reduction is too small, then the hypothesis is rejected and the sample becomes a key sample.

Alike to the off-line inclusion, the significance ζ is not considered as an extra parameter that has to be given a value, because the effect it has on the inclusion of a sample can also be obtained by altering the noise level.

With the inclusion, we assume for the sake of implementation in appendix B, that the new key sample enriches the structure, so that its target can be approximated:

Assumption 5 (Structure enhancement) *When a sample becomes a key sample, the structure is enhanced such that the new sample can be exactly approximated with the new structure.*

Omitting a key sample

If a key sample becomes superfluous it can be omitted to save memory space as well as computation time. This is of special interest in the on-line setting because the calculations have to be done within a sample period. Furthermore, fewer parameters result in less uncertainty of the remaining parameters. A two-step approach is used to find the key samples that can be removed:

1) Find the key sample that gives the smallest error increment after omission.
2) Test whether this error increase is acceptable.

This two-step approach is decided on, because the second step takes considerable computation time. By this two-step approach, these computations only have to be performed for the key sample that gives the smallest error increment. First the key sample that will cause the smallest error increase will be located.

In Hassibi and Stork (1993) a pruning method called Optimal Brain Surgeon (OBS) is introduced. In their paper the change in the summed squared approximation error is calculated as a function of the parameter change. These calculations can be used to calculate the error change if an element of the parameter vector is set at zero. The error change as a function of the parameter change is approximated by a Taylor series:

$$\Delta E = \left(\frac{\partial E}{\partial \mathbf{b}}\right)^{\mathrm{T}} \Delta \mathbf{b} + \tfrac{1}{2}\Delta \mathbf{b}^{\mathrm{T}} \mathbf{H} \Delta \mathbf{b} + O(\|\Delta \mathbf{b}\|^3). \tag{4.56}$$

In this equation, $\mathbf{H}$ is the Hessian matrix of the summed squared approximation error with respect to the parameters. Hassibi and Stork (1993) assumed that the approximation is in a (local) minimum, so that the first term on the right hand side equals zero. Furthermore, the contributions of the third- and higher-order terms are neglected. This results in:

$$\Delta E = \tfrac{1}{2}\Delta \mathbf{b}^{\mathrm{T}} \mathbf{H} \Delta \mathbf{b}. \tag{4.57}$$

Equating one of the elements of the parameter vector with zero, is the same as omitting it. This can be expressed as:

$$\mathbf{1}_i^{\mathrm{T}} \Delta \mathbf{b} + b_i = 0, \tag{4.58}$$

in which $\mathbf{1}_i$ denotes the identity vector with a 1 at the i^{th} location and all other terms equal to zero.

We want to find the difference for the parameter vector that minimally increases the error, under the condition that one parameter element becomes zero. This is done by forming the Lagrangian for the minimisation of (4.57) with constraint (4.58):

$$\mathcal{L} = \tfrac{1}{2}\mathbf{\Delta b}^{\mathrm{T}}\mathbf{H}\mathbf{\Delta b} + \alpha(\mathbf{1}_i^{\mathrm{T}}\mathbf{\Delta b} + b_i). \tag{4.59}$$

Differentiation of this Lagrangian with respect to $\mathbf{\Delta b}$ and α yields:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{\Delta b}} = \mathbf{H}\mathbf{\Delta b} + \alpha \mathbf{1}_i = 0, \tag{4.60a}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = \mathbf{1}_i^{\mathrm{T}}\mathbf{\Delta b} + b_i = 0. \tag{4.60b}$$

This can be written in matrix form:

$$\begin{bmatrix} \mathbf{H} & \mathbf{1}_i \\ \mathbf{1}_i^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{\Delta b} \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ -b_i \end{bmatrix}. \tag{4.61}$$

The block matrix inversion lemma (Kailath, 1980) is used to obtain the following results:

$$\mathbf{\Delta b} = \frac{b_i \mathbf{H}^{-1}\mathbf{1}_i}{[\mathbf{H}^{-1}]_{ii}}, \quad \Delta E = \frac{b_i^2}{2\,[\mathbf{H}^{-1}]_{ii}}. \tag{4.62}$$

The computational load to calculate the inverse of the Hessian as well as the Hessian itself, is considerable.

A simplification of this scheme is given in Cun et al. (1990) in a scheme known as Optimal Brain Damage (OBD) in order to avoid the calculation of the inverse. This scheme assumes that the Hessian is a diagonal matrix which simplifies the calculations for the error increment:

$$\Delta E \approx \tfrac{1}{2} b_i^2\,[\mathbf{H}]_{ii}\,. \tag{4.63}$$

Also if the Hessian is diagonal dominant, this is a reasonable approximation.

Both OBD and OBS were constructed for pruning in artificial neural networks. This means that the first derivative of the error surface with respect to the parameters is only zero if the parameters in the network have converged. In a LS or RLS setting, the weights have always converged, and therefore these schemes can be used in an on-line fashion (Leung, Wong, Sum, and Chan, 2001). This makes them a good starting-point to use it as a pruning scheme.

The Hessian for the least squares setting is straightforwardly calculated. The approximation error is given as:

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\mathbf{b}, \tag{4.64}$$

from which the summed squared error is derived:

$$\mathbf{e}^{\mathrm{T}}\mathbf{e} = \left(\mathbf{y}^{\mathrm{T}} - \mathbf{b}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\right)\left(\mathbf{y} - \mathbf{X}\mathbf{b}\right), \tag{4.65a}$$

$$= \mathbf{y}^{\mathrm{T}}\mathbf{y} + \mathbf{b}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{b} - 2\mathbf{y}^{\mathrm{T}}\mathbf{X}\mathbf{b}. \tag{4.65b}$$

Differentiation with respect to the parameter vector yields:

$$\frac{\partial \mathbf{e}^{\mathrm{T}}\mathbf{e}}{\partial \mathbf{b}} = -2\mathbf{X}^{\mathrm{T}}\mathbf{y} + 2\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{b}, \tag{4.66a}$$

$$\frac{\partial^2 \mathbf{e}^{\mathrm{T}}\mathbf{e}}{\partial \mathbf{b}^2} = 2\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{H}. \tag{4.66b}$$

The first equation (4.66a) is equal to zero due to the normal equations, while the second gives the Hessian. Further differentiation would result in a zero vector, which makes the error change of (4.57) exact in the least squares setting. Including the Hessian of (4.66b) into (4.62), gives:

$$\Delta E = \frac{b_i^2}{4[(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}]_{ii}}. \tag{4.67}$$

An interesting interpretation of this equation is found if it is noted that the term $\sigma^2(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ is the covariance matrix of the parameter vector (Draper and Smith, 1998). The parameter value divided by its uncertainty is proportional to the test whether the hypothesis '$b_i = 0$' can be rejected. If this hypothesis is considered the null hypothesis, then it is rejected with a significance of ζ if:

$$P\left(\frac{b_i^2}{\sigma^2[(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}]_{ii}} > z_\zeta\right) < \zeta. \tag{4.68}$$

Here, z_ζ and ζ are coupled by a χ^2-distribution. So, omitting the parameter that has the smallest error increment is the same as omitting the parameter which is most likely to be zero.

Instead of dividing the parameter value by the uncertainty of the parameter, a second possibility to select a candidate for omission, is to divide the parameter value by the uncertainty of the key sample:

$$i = \arg\min_i \frac{b_i^2}{[\mathbf{V}]_{ii}} \quad i = 1 \ldots k, \tag{4.69}$$

in which i becomes the candidate key sample to be omitted. Although this ratio is no longer directly related to the error increment, it has the intuitive interpretation that the key sample which has not much influence and which only represents a small number of samples, can be omitted.

Furthermore, this alternative test is motivated by reasons of both numerical stability and calculation speed. If there are highly correlated indicators, then the condition number of the matrix $\mathbf{X}^T\mathbf{X}$ is bad. This makes division by the inverse of the diagonal elements prone to numerical problems. The certainty of the key samples is not explicitly determined by the values of the indicator vectors, but by the samples that they represent. Therefore, nearly dependent indicators do not necessarily pose a problem. Experimental results confirm this. For an arbitrary data set of 5 000 training samples, the condition number of $\mathbf{X}^T\mathbf{X}$ is $\kappa \approx 10^{17}$ while for $\mathbf{V}$ it is around 100.

The second reason to use the variance of the key sample is due to the calculation speed. The covariance matrix tends to be diagonal dominant, especially if limited samples are supplied. Therefore, the same assumption can be made as was made for OBD, giving the following selection:

$$i = \arg\min_i b_i^2 [\mathbf{V}^{-1}]_{ii}. \tag{4.70}$$

This minimisation can be done fast, because $\mathbf{V}^{-1}$ is known.

However, apart from the calculation speed and the numerical stability, it is of course important that a *correct* key sample is selected for omission. Or at least a key sample that gives an error increment that is near the minimal error increment. Experiments that used the selection mechanism of (4.70) showed that key samples were selected that gave an error increment close to the minimum. This is illustrated by the next example.

Example 4.6 (Selection of candidate for removal)
A piecewise linear approximation of data set 1 is made by KSM. After the approximation, the different omission schemes are compared. Refer to figure 4.4. The black dots correspond to the error increment calculated by the OBS scheme for the different key samples. The dark gray dots denote the approximate error increment found by the OBD omission scheme. The light gray dots correspond to the arguments of (4.69) and (4.70), which are nearly on top of each other. For ease of reference, these selection mechanisms are called RKSM $^{\text{OBS}}$ and RKSM $^{\text{OBD}}$ respectively. Based on this figure several remarks can be made:

- The approximation of RKSM $^{\text{OBS}}$ by RKSM $^{\text{OBD}}$ looks good. The lines nearly cover each other. *For this example* the matrix $\mathbf{V}^{-1}$ is diagonally dominant.

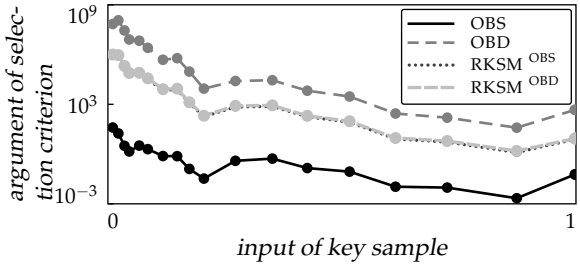


Figure 4.4: Importance calculated by different pruning schemes

- The OBD should give an approximate of the error increment. But in this example the difference between OBS and OBD is significant. Inspection of the Hessian shows that it is not diagonal at all.
- All the selection mechanisms find the same candidate key sample.

▩

Because of the small computational load, the numerical stability and the observations that it selects good omission candidates, the selection criterion of (4.70) is used from now on.

The location of the candidate key sample was the first step of the two-step approach. By the use of (4.70), the increase in the error is no longer known. However, in the implementation given in appendix B, the error increase is available just before a key sample is omitted. It can be tested whether this error crosses some bound before the key sample is actually omitted.

4.2.4 Some tools

There are some techniques that are often used in practise and that can be incorporated into this scheme.

Forgetting

If the behaviour of the supervisor is time-variant, the samples that were supplied to the approximator in the past lose significance. To incorporate this, a forgetting mechanism can be incorporated into the approximator, see e.g. (Haykin, 2002). This forgetting means that old samples are of less significance than new samples. In RKSM this result can be achieved by multiplying the matrix $\mathbf{V}^{-1}$ with a constant between zero

and one. The prediction is not altered due to this, but due to the decreased weight, the prediction can more easily be adapted by new samples.

When the function is changing too fast for the forgetting factor used, problems arise. Instead of altering the parameters in this case, the structure is modified. This happens because the new sample is thought of as an unlikely occurrence for the given structure, resulting in a non-zero parameter, i.e. a structure adaption. The use of the forgetting mechanism is illustrated in section 4.3.5.

Even if the function is not time-variant, a (time-dependent) forgetting factor is a good method, to indicate that the structure in the beginning of the approximation is premature, and should not get too much significance. As key samples selected in the beginning of the approximation would represent a large multitude of data, its weight is considerable. However, due to the premature structure in the beginning, the targets of this key sample might contain flaws. Introducing a (time-dependent) forgetting factor acknowledges that this premature structure might not be correct, and is therefore uncertain. In Schaal, Atkeson, and Vijayakumar (2002) time-dependent forgetting is also used to account for the incorrect structure in the beginning of the approximation.

Regularisation

The ridge regression scheme (Hoerl and Kennard, 1970) can be applied in this setting without problems. This regularisation scheme minimises the norm of the parameter vector. In section 2.1 it has been argued that this is the same as including the a priori knowledge that the parameters are likely to be zero. The regularisation parameter determines how certain one can be of this.

Whenever a new key sample is included, and therefore a new parameter, the regularisation has to be applied to this parameter. This can be done by introducing an *extra* sample to the approximator that has the same input as the new key sample, but the target of which corresponds to the current prediction. This makes the parameter of the new key sample go in the direction of zero, just as done by ridge regression. The weight of this regularisation sample can be equated with the regularisation parameter.

(Near) singularity

The inclusion of a key sample of which a copy is already there, makes the matrix $\mathbf{X}_{\text{ks}}$ singular and should therefore not be done. Also, if a

key sample is included that is close to another key sample in the feature space, this would make the matrix $\mathbf{X}_{ks}$ nearly singular. This gives rise to errors in the parameter vector. Therefore, next to testing whether a key sample should be included because the current structure cannot be correct, we should also test if the inclusion significantly improves the structure.

To test whether the new key sample is significantly different from previous key samples, the residual after the projection of the new dual indicator vector on the current indicator vectors, can be investigated. If this residual is too small, it means that the indicator vector induced by the new key sample, can be nearly written as a linear combination of the present indicator vectors. It is user-specified when the residual is considered too small. However, if it gets near the numerical precision of the computer, it is certainly too small. This test is a byproduct of the addition of the key sample to the set of key samples and does not take any time. The implementation can be found in appendix B.

The test for significant contribution can also be used to limit the number of key samples by not including key samples with a small residual. This is illustrated in section 4.3.4.

Next to the test if the inclusion of the key sample makes the indicator matrix of key samples $\mathbf{X}_{ks}$ nearly singular, a test should be done to see if the matrix $\mathbf{X}_{ks}^T \mathbf{V}^{-1} \mathbf{X}_{ks}$ becomes nearly singular by the addition of the new key sample. This test checks whether the new key sample supplies enough information to the weighted key samples.

The complete algorithm for RKSM is given in algorithm 4.2. The objective of this algorithm is not to show all the necessary calculations, but to show how the different parts are put together. For the implementation details, see appendix B.

4.3 Evaluation

Before the introduced function approximator is compared with other methods, some of its properties are evaluated. Different aspects of the approximator are treated.

4.3.1 Information extraction

The goal of the first set of function approximations is to test the ability of the on-line function approximator to extract the required information from a data stream.

Algorithm 4.2: Recursive Key Sample Machine algorithm

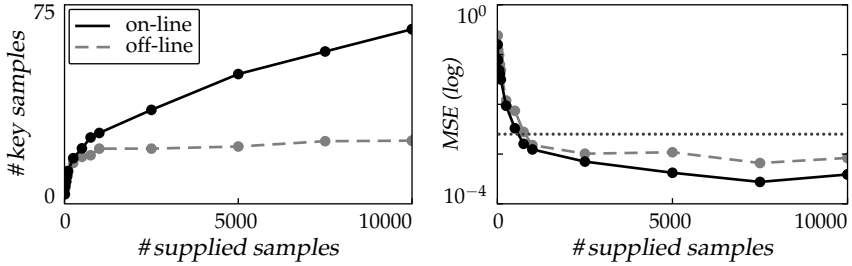
Given: $\mathbf{x}_{ks}, \mathbf{X}_{ks}, \mathbf{y}_{ks}, \mathbf{V}^{-1}$ and a new sample $(\mathbf{x}, y)$
1: form **f** based on **x**
2: calculate prediction for this input
3: **if** parameter unlikely to be zero (4.55) **then**
4: include as key sample (algorithm 4.1(b))
5: **else**
6: fine tune key samples (algorithm 4.1(a))
7: **end if**
8: find candidate for omission (4.70)
9: calculate error increment (4.67)
10: **if** error increment is small enough (pg. 172) **then**
11: omit key sample (algorithm 4.1(c))
12: **end if**

This is tested by offering different amounts of data to the approximator and investigating the approximation error as well as the number of key samples required. The results are compared with the results of the off-line approximation. This comparison is made to see if the on-line approximator can extract the same information from the data stream as the off-line approximator can from the complete data set. Three cases are considered:

Case 1) The significance level for testing the hypothesis whether a key sample should be included, is the same for the on-line and off-line approximator as is the noise level. This situation is created in order to investigate the differences between the inclusion schemes for key samples. The inclusion scheme determines which samples become key samples, and is therefore of importance to understand the differences between the on- and off-line approximators.

Case 2) In order to test whether the on-line method extracts the correct key samples from the stream, the number of key samples is made the same as the number of key samples in the off-line approximation by altering the significance level. Altering the noise level instead would have had the same effect.

Case 3) The significance levels of on- and off-line approximation are



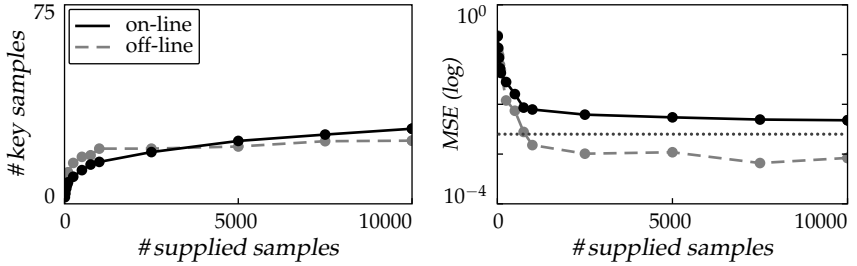
(a) The number of key samples (b) MSE of validation set

Figure 4.5: Case 1: on-line and off-line approximation with equal inclusion significance, no pruning

equated again. The number of key samples for the on-line approximation are now equated to the number of key samples for the off-line approximation by the pruning mechanism. The allowed error before a key sample is omitted is selected so that the number of found key samples becomes equal for both methods.

The first case sets the significance for the rejection of the hypothesis, quite arbitrarily, for both at 8%. The estimated noise variance is equated with the true noise variance which is $(0.05)^2$ and the approximation is made by a piecewise linear approximation. The results are averaged over ten approximations. The validation error is calculated, based on 10 000 noiseless samples. Different numbers of training samples were inputted to the off-line approximator for which the performance was determined. The performance of the on-line approximator was tested after it processed this number of samples. The average result for the approximations for different amounts of data is given in figure 4.5.

The first thing that stands out is the difference of key samples between on- and off-line approximation. In the on-line approximation the number of key samples keeps growing fast and more key samples are found in the end. The number of key samples for the off-line approximation grows slowly. This behaviour can be explained by the inclusion mechanism of the approximators: both methods use a statistical test to find out if a sample should becomes a key sample. In the off-line approximation the test checks if the inclusion of the sample will have a statistically significant error reduction *of the complete set of samples.* However, the inclusion of a key sample in the on-line approximation is based on the weighted change in the key samples *plus* the residual of the newly supplied sample. If the residual of the new sample is too



(a) The number of keysamples (b) MSE of validation set

Figure 4.6: Case 2: on-line and off-line approximation with approximately equal key samples, no pruning

large, it is included. As a result of this inclusion scheme, samples with a large realisation of noise will be included as key samples. This will result in a growth of the number of parameters if more samples are supplied, as observed in figure 4.5(a). The fast growth of the parameters is not acceptable when applied in a control setting, because of the limited resources. Therefore, two mechanisms are tested hereafter to limit this growth: adapting the significance level with fixed noise level, and pruning of superfluous key samples.

Although it might be expected that the approximation would be better for the off-line approximation, it is found in figure 4.5(b) that the on-line approximation has a smaller validation error. This can be accounted for because we see that the number of key samples in the on-line approximation is much larger. With these extra key samples, more freedom is obtained for the approximation and therefore, the comparison is not fair. In the next approximation, the number of key samples is the same, to test which approximator selects a better set of key samples.

The real danger of including more key samples is overfitting. If more and more key samples are included, one might start fitting noise. The MSE on the validation set for the on-line approximation rises slightly for a large number of supplied samples, which can suggest that overfitting occurs.

In these approximations, both methods converge rapidly to an approximation with a validation MSE below the variance of the noise, given by the dotted line.

In order to test whether the on-line method can extract the correct

key samples from the stream, the number of key samples is approximately made the same as the number of key samples in the off-line approximation by altering the significance level. The noise level is kept fixed. The validation errors and the numbers of key samples found for this approximation are given in figure 4.6. Only the on-line approximation is altered.

It is clear that if both methods can select the same number of key samples, that the off-line method gives a smaller validation error. This is as expected, because it can use all the data in the selection scheme.

The MSE on the validation set is no longer smaller than the variance of the noise. Furthermore, it can be observed that the growth of the number of key samples is much slower. The same reason as before, large noise realisations, make the growth continue.

In the third case, the significance of the hypothesis testing for the on-line approximation is again set at 8%. However, the pruning mechanism is activated in this set of approximations. It omits those key samples that have a limited influence on the approximation. The acceptable error increment when omitting a key sample is chosen in such a way that approximately the same number of key samples is found as for the off-line approximation.

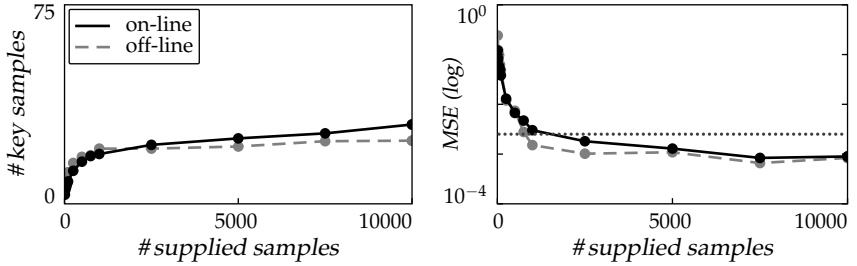
The results of the approximation with the pruning are given in figure 4.7. Although the off-line approximation still has a smaller MSE for the validation set the performance of the on-line approximation is good. This shows that by omitting the useless key samples, a good set of key samples is found. Both the on- and off-line approximator come to a approximation of which the MSE is smaller than the variance of the noise.

Still a small growth is observed. It is expected that the number of key samples stabilises if more samples are supplied, because the key samples might be included due to a large noise realisation, but if they are not required for the structure, they will be omitted in time.

4.3.2 Noise level

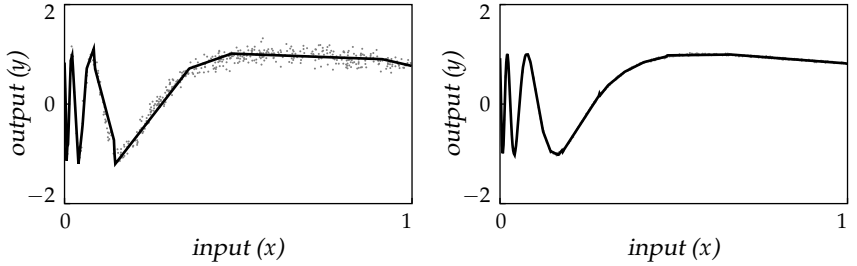
Although the comparison with respect to the off-line setting shows that the on-line approximator is capable of extracting correct key samples from a data stream, further tests have been performed to show explicitly some of its properties.

In this set of approximations 5 000 data samples from data set 1 are supplied to the approximator with different noise variances. This data is approximated by a piecewise linear function. Within this set of approximations, the noise level is assumed to be known, so that we can



(a) The number of keysamples (b) MSE of validation set

Figure 4.7: Case 3: on-line and off-line approximation with key sample pruning



(a) $\sigma_n = 0.1$, # key samples = 20 (b) $\sigma_n = 0.01$, # key samples = 38

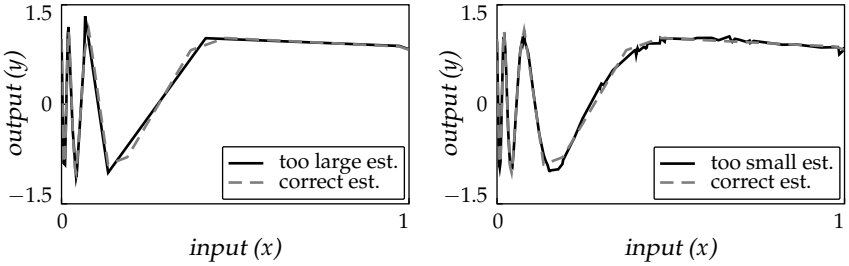
Figure 4.8: Approximation of RKSM for different noise levels

concentrate on how the method handles different noise levels. The sensitivity to the estimate of the noise level will be treated next.

The approximation results for different noise levels are given in figure 4.8. The standard deviation of the noise is given for each figure, as well as the resulting number of key samples. Based on these figures, the following remarks can be made. For this and coming function approximations, the significance level is set to 8%.

First, if there is less noise, a better approximation can be made. The less noise there is on the targets, the more key samples are included by the selection scheme and therefore a more precise approximation is obtained. More key samples *can* be included because the good quality of the data makes the approximator less prone to fit noise.

Second, by adjusting the precision of the approximation to the noise level, only few key samples are found for large noise levels. As a result, the method is not prone to overfit the data.



(a) Approximate with too large noise estimate

(b) Approximate with too small noise estimate

Figure 4.9: Sensitivity of RKSM for different noise estimates

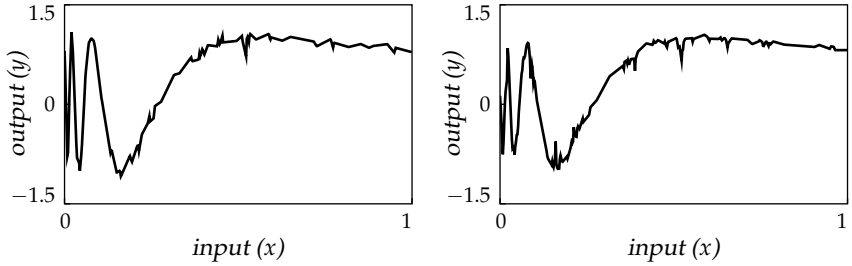
The last observation that can be made is that the fast fluctuations for small input values are approximated, even if the noise is significant. This is because these fluctuations are distinguishable from the noise for the noise levels given.

4.3.3 Sensitivity for noise estimate

The noise variance is often unknown and therefore an estimate has to be made use of. The effect of an incorrect estimate of the approximation is shown in figure 4.9. In (a) the estimated standard deviation is twice the real standard deviation while in (b) the estimated standard deviation is half the true noise standard deviation. As in the off-line approximation, this estimate determines the precision by which the data is approximated. If the noise is overestimated, the approximation is rough, if underestimated, too many key samples are added, resulting in a noisy approximation.

4.3.4 Regularisation

In the previous test it was shown that if the noise was underestimated, too many key samples are included, resulting in a noisy approximation as shown in 4.9(b). In this figure, the true noise level is 0.05 and the estimated noise level is 0.025. By assuming that the approximation is smooth (assumption 3), we know that this noisy approximation is incorrect. In order to obtain a smoother approximation, the ridge regression scheme is used. The result of this regularisation is given in figure 4.10(b). Directing the parameter to zero, does not have a smoothing effect on



(a) No regularisation ($\lambda = 0.0$) (b) With regularisation ($\lambda = 1.0$)

Figure 4.10: Approximation without and with regularisation for too small noise estimate

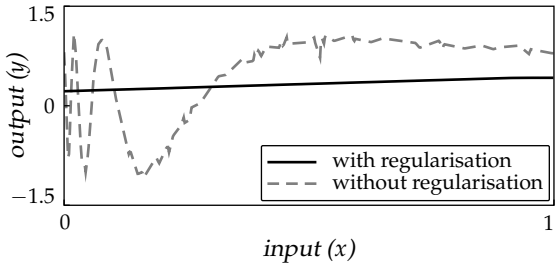


Figure 4.11: Approximation with large regularisation parameter ($\lambda = 10$)

the approximation as can be seen in the figure. The approximation even worsens!

Investigating what happens if the value of regularisation parameter is selected large ($\lambda = 10$) gives insight in why this approximation is not smoothed by the ridge regression scheme. The approximation with the large regularisation parameter is given in figure 4.11. In this approximation nearly all the offered samples become key samples, while the approximation of the data stays the same after the inclusion of the first key sample.

In this approximation the first key sample is included without regularisation because it is unknown in what direction we like to pull it. This approximation is not yet a good approximation of the data. A following sample is therefore not predicted well, and the inclusion mechanism includes this sample as a key sample. However, due to the large regularisation, the target of this key sample is made (nearly) the same as the current prediction *and the prediction does not change.* This is in contradiction with assumption 5. So, future data is not approximated any better.

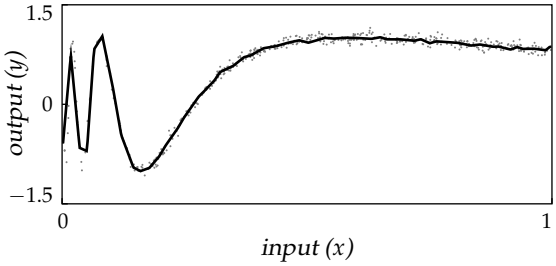


Figure 4.12: Structural regularisation

This inclusion continues for future samples and because nearly all samples are included as key samples, no samples are left to alter the key samples.

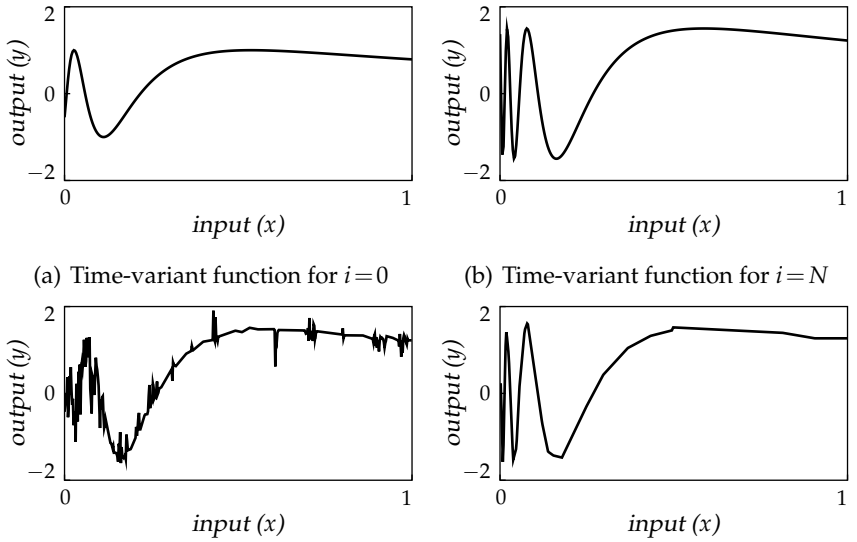
If the regularisation parameter is given a smaller value, a similar behaviour is found, though less extreme. The key sample's target is made to approach towards the previous approximation, so that it does not lead to the required improvement. Therefore, more samples are included because of the large error. This is the effect underlying the negative influence of this form of regularisation in figure 4.10: so, the ridge regression scheme is not a good option for this approximation scheme.

An alternative regularisation scheme acts on the *structure.* This way of regularisation allows a sample to become a key sample *only* if it passes the selection criterion (4.55), *and* if it contains sufficient information. The scheme that was introduced in the previous section to avoid near singularities can be used to accomplish this. The result of this regularisation form is illustrated in figure 4.12. The approximation for figure 4.12 uses approximately a third of the key samples that were required for figure 4.10. Due to the use of fewer key samples, the approximator becomes less noisy. However, the danger of this technique is that fast fluctuations can no longer be approximated, as seen for small input values.

4.3.5 Time variant functions

In order to test the ability to approximate a time-variant function, a new data set is constructed. In this data set the sample number is of influence on the relation between the in- and output. The function from which samples are drawn is given as:

$$y_i = \left(1 + \frac{i}{2N}\right) \sin\left(\frac{1}{x_i + 0.1\left(1 - \frac{i}{2N}\right)}\right) + \varepsilon. \tag{4.71}$$



(a) Time-variant function for $i = 0$

(b) Time-variant function for $i = N$

(c) Approximation of time-variant function *without* forgetting

(d) Approximation of time variant function *with* forgetting

Figure 4.13: Approximation of a time-variant function

In this equation N is the number of samples and i is the sample number. For this evaluation, $N = 10\,000$. This function is plotted in figure 4.13(a) for $i = 0$ and in figure 4.13(b) for $i = N$. x_i is generated with a uniform distribution between zero and one and is independent of the sample number. The output is corrupted by additive Gaussian noise with a standard deviation of 0.05, denoted by ε.

The data is approximated with and without forgetting. The forgetting factor was set at $5 \cdot 10^{-3}$, which was found found after a few tries. The results of these approximations are shown in figure 4.13. It can be seen that the approximation scheme with forgetting is able to follow the time variant function so as to find the final function, while the scheme without forgetting tends to accept old samples of equal importance, giving an incorrect approximation.

4.3.6 High-dimensional input

The last test that is performed is to see how RKSM reacts when higher-dimensional data is provided. The data is generated from the function

Table 4.2: Growth of number of key samples due to increase of input dimension

input dim.	# key samples	MSE
1	16	$2.05 \cdot 10^{-4}$
2	29	$2.74 \cdot 10^{-4}$
3	47	$3.51 \cdot 10^{-4}$
4	66	$6.34 \cdot 10^{-4}$
5	89	$7.72 \cdot 10^{-4}$

introduced in section 3.5.2:

$$y(\mathbf{x}) = \frac{1}{n_{\text{inp}}} \sum_{i=1}^{n_{\text{inp}}} 2 \max(\sin(3\pi \mathbf{x}_i) - \tfrac{1}{2}, 0). \tag{4.72}$$

In this equation the subscript of $\mathbf{x}$ denotes the input number. Although this function is an additive combination of one-dimensional functions, the RKSM does not have this information and approximates the function as a n_{inp}-dimensional function.

The approximation is made for different input dimensions. For each approximation 20 000 samples were supplied to the approximator and the results presented in table 4.2 are the average of five runs. Although there is no noise corrupting the targets, the noise estimate is given some value. If this was not done, all samples would become key samples due to the division by zero in (4.55). The pruning mechanism is used, as well as the forgetting mechanism. The forgetting mechanism is used to enforce some uncertainty to the preliminary structure.

From this table it is clear, that if the input dimension is increased, the number of key samples grows slowly. To get a similar MSE as found in table 4.2, a similar number of key samples is required in the off-line approximation. In figure 4.7 it was also found that the number of key samples is similar for the on- and off-line approximation, so it is expected that they handle higher-dimensional input spaces similar.

The MSE of the validation set grows slowly. This can be explained because the same amount of data is used to train a larger number of key samples. Therefore, the variance of these key samples is larger (Draper and Smith, 1998).

4.3.7 Some notes on computation time

To get a notion of the time that is required for an approximation, 10^6 training samples from data set 1 are approximated with different noise estimates. Due to the different noise estimates, different numbers of key samples result. The computation time required is given in figure 4.14. As with the off-line calculation time, the code used was developed to test the ideas and has not been optimised for speed.

In this figure, it can be seen that the calculation time grows approximately linearly with the number of samples processed. The slight above-linear growth can be explained by the inclusion of more key samples if the number of processed samples increases. The calculation time for 10^6 samples with 110 key samples is approximately 90 [s]. This gives an average calculation time per processed sample of 90 [µs], which is well within the sampling period of a controlled mechanical system of our interest.

4.4 Comparison

The evaluations of the previous section showed the general properties of the RKSM, but it is interesting to see how this method functions when compared with other on-line structure-adapting function approximators. The number of function approximators that recursively adapts the parameter as well as the structure is fairly small. Most of the methods that *are* capable of this, are the so-called 'lazy'-methods. They are called lazy because they store all the data and do not make a prediction until a prediction is required (Aha, 1997; Atkenson, Moore, and Schaal, 1997; Uysal and Güvenir, 1999). However, because they store all the data, they cannot be used in a control setting in which an endless stream of data is presented.

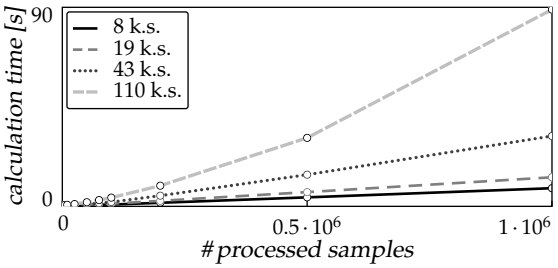


Figure 4.14: Computation time as a function of the number of samples processed. Different number of noise estimates are used.

The only method found that does not store all the data and alters its structure as well as its parameters is Locally Weighted Partial Regression (LWPR) given in (Schaal et al., 2002). This method is a continuation of previous work of the same authors, see e.g. (Conradt, Tevatia, Vijayakumar, and Schaal, 2000; Schaal and Atkeson, 1998; Vijayakumar and Schaal, 2000). Because this method is only used for comparison reasons, the method will be treated concisely.

4.4.1 Locally Weighted Partial Regression

The idea of LWPR is to make a set of low-order models which are responsible for the approximation of the data in a limited region of the input space only. The parameters of these local models are updated by a recursive least squares (RLS) scheme. The contribution of each local model to the prediction is determined by an activation function, e.g. a Gaussian function. The weighted sum of the outputs of the local models determines the actual prediction. This activation level also determines the level of adaption of the local model when a new sample is supplied to the approximator.

The activation function of the local models as well as the parameters of the local models are altered by the supplied data. In the case of the Gaussian activation functions, the spread matrix — a spread matrix is used because we are working in an n-dimensional space — is altered by the samples to indicate the region of activity. The result of this is that in regions where more local models are needed to approximate the data, more models *will* be present. Furthermore, if a low-order model can describe the data in a large part of the space, only one model will describe the data in that part of space.

The adaptation of the activation region for the local models is done with a gradient descent method. Therefore, the convergence of regions is not as fast as the convergence of the parameters of the local models. In order not to converge to delta-like activation functions at training samples, a cross-validation scheme is used in the update.

Due to the alteration of the regions, it might happen that a region of the input space is not covered by a local model, or that too many local models cover it. If a new sample is supplied to the method and it is found that the summed activation of all the local models at that location is below some threshold, a new local model is inserted. The activation region of the new model is centred at the location of this sample, and the spread is set at some predefined value. The value of the new spread has a significant influence on the behaviour of the approximator.

Next to the detection of too little activation in a certain region, it is also tested whether there is too much activation in certain regions. If so, a pruning mechanism is activated in order to save memory and computation time.

An advantage of the use of the local low-order models, is that there is an abundance of literature on this topic and the knowledge developed for least squares can be used. In (Schaal et al., 2002) the projection pursuit idea is incorporated in this scheme, making it applicable for use in high dimensional spaces.

4.4.2 Data extraction

Data set 1

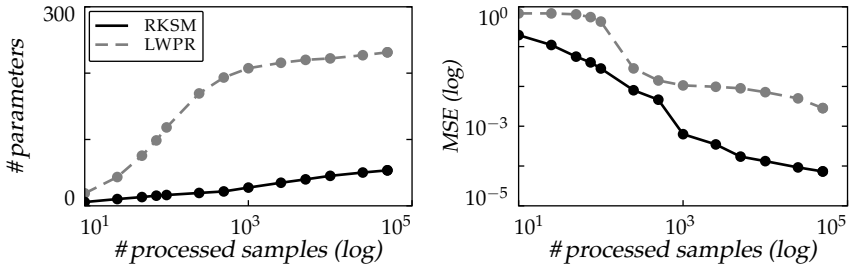
The first set of function approximations is on data extraction. This tests whether the methods are capable of finding the specific relation underlying the data. The two data sets introduced in chapter 1 are used for this comparison. As with the comparison with the off-line approximation, the number of supplied samples is increased and the number of required parameters and the validation error are recorded. The data of example one is corrupted with noise with a standard deviation of 0.01.

Ridge regression is not used in either method, nor is there a limit set to the closeness of the key samples or the equivalent measure in LWPR. Because the structure is gradually built up, a preliminary structure is found in the beginning. A forgetting factor is used to prevent this initial structure from having too strong an effect on the final approximation.

The RKSM approximates the data by a piecewise linear function. The correct noise variance is used. The LWPR locally uses a linear approximation which is weighted by a Gaussian kernel function so that a smooth function is obtained. No noise estimate is required for LWPR.

The result averaged over ten runs is depicted in figure 4.15. Please note that all axes are in logarithmic scale except the number of parameters. In (a) the number of parameters is given. For each local model the LWPR needs four parameters: centre and spread of the activation function; bias and slope of the local model. For each key sample in RKSM, only two parameters are needed: the input value of the key sample and its target. This difference is incorporated in the figure. The black line is used for RKSM while the gray dashed line is for LWPR.

In this example the RKSM outperforms the LWPR. The MSE on the validation set is 5 to 50 times smaller while only one fifth of the parameters is required. The LWPR has great difficulties in finding the correct



(a) # key samples (b) MSE of validation set

Figure 4.15: Comparison for RKSM and LWPR on data of data set 1

Table 4.3: MSE and number of parameters on data set 2

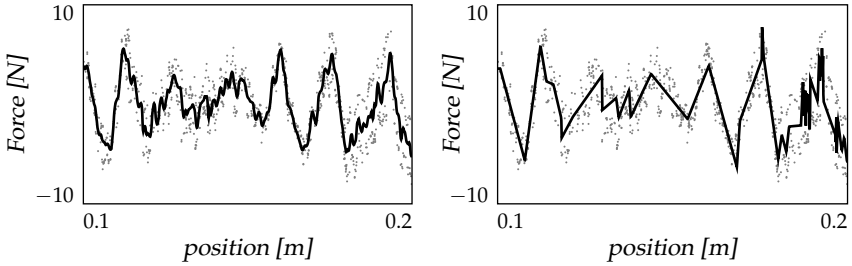
LWPR			RKSM		
D_{init}	MSE	# param	σ_{est}	MSE	# param
10^3	3.7	160	1	4.0	42
10^4	3.4	520	0.8	3.9	137
10^5	3.2	1 560	0.5	4.2	396

structure for the fast fluctuations. This is explained by the iterative refinement of the region in which each local model is active.

Data set 2

The second test on data extraction is performed on the data of example two. The first 10 000 data samples were used to train the approximators, while the last 5 000 were used to calculate the validation error. No regularisation is applied. The results of these approximations are given in table 4.3, while the actual approximation for the smallest MSE on the validation set is plotted in figure 4.16. In the table D_{init} stands for the initial size of the created regions.

It shows clearly in 4.16(b) that for position values around 0.19 [m], the approximation of RKSM fluctuates swiftly to approximate the samples at both sides of the 'gap.' This behaviour is due to the addition of a key sample whenever the weighted key sample change plus the residual of the new sample is too large for the current structure. If due to some effect other than noise, the targets show some 'gap', then the residual of the new sample will be large, resulting in the inclusion of it to the set of key samples. Although some of these key samples are omitted



(a) Approximation by LWPR (b) Approximation by RKSM

Figure 4.16: Approximation of data set 2

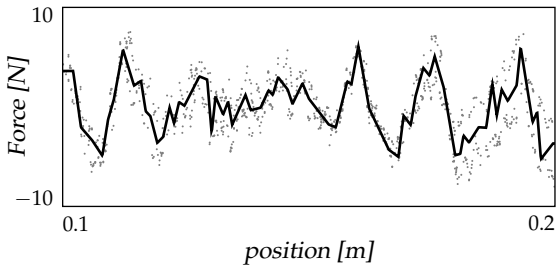


Figure 4.17: Approximation with structural regularisation

by the pruning mechanism, wild jumps are found in the approximation. Increasing the noise estimate avoids this behaviour, but this also limits the precision by which the data can be approximated.

The performance of the approximation by RKSM can be increased by applying structural regularisation. Because of the regularisation, the noise estimate can be decreased so that a more accurate approximation is found. The before-mentioned fluctuations will not occur, because key samples too close to each other are not allowed. With regularisation the noise estimate could be chosen much smaller than before as $\sigma_{\text{est}} = 0.4$. The MSE on the validation set was found to be 3.5 with a total of 130 parameters. This approximation is shown in figure 4.17. Note that in the data there *are* fluctuations at 0.19 [m] so that these should be approximated.

The approximation with LWPR gives a marginally smaller MSE on the validation set using many more parameters. LWPR forms a linear approximation locally and is therefore less hindered by the dual targets. The fluctuations that are shown in 4.16(a) are less if the initial size of the regions is larger, but this also results in a larger validation error.

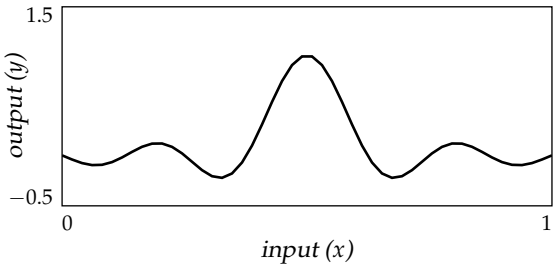


Figure 4.18: A sinc function

Although the RKSM is capable of handling the double target, it can only do this by incorporating a form of structural regularisation. LWPR can inherently cope with them.

4.4.3 Noise handling

Next it is tested how well these methods handle the noise corrupting the targets. The RKSM uses the noise level explicitly in order to avoid overfitting and it is therefore expected that large noise variances will give bigger validation errors. In contrast with this, the LWPR does not use the noise level explicitly. LWPR adapts the regions of activation based on leave-one-out cross-validation and the noise variance is expected not to have a large influence on the approximation.

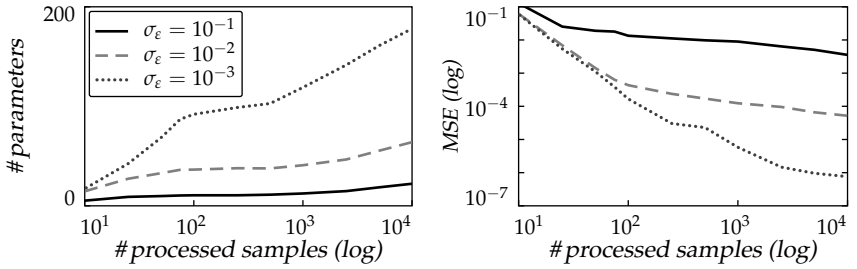
Because the LWPR had difficulties in finding the data generated by the function of data set 1, a slowly fluctuating sinc-function is used for this experiment:

$$y_i = \frac{\sin(25x_i - 12.5)}{25x_i - 12.5} + \varepsilon. \tag{4.73}$$

The input samples x_i are taken from a uniform distribution between zero and one. The sinc-function is plotted in figure 4.18.

The variance of the noise is varied in this set of approximations to see its effect. The MSE as well as the number of parameters are recorded as a function of the number of samples offered. The RKSM approximates the data by a piecewise linear function. The correct noise estimate is used.

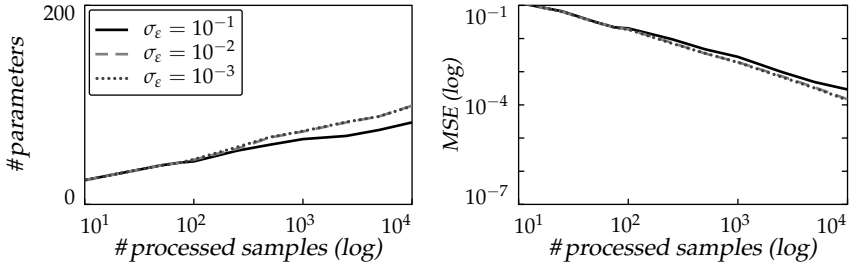
In LWPR the initial size of the local models is changed so that a good estimate is found within the 10 000 samples that were presented to both methods. The results are averaged over ten runs.



(a) Number of required parameters for piecewise linear approximation

(b) MSE of validation set for piecewise linear approximation

Figure 4.19: Influence of different noise levels for RKSM



(a) The number required parameters

(b) MSE of validation set

Figure 4.20: Influence of different noise levels for LWPR

In figure 4.19 the results for RKSM are shown, and in figure 4.20 for LWPR. As expected, the validation error of RKSM depends on the noise level and larger noise levels result in larger validation errors.

The results of LWPR show that the noise level is of minimal influence on the validation error. The validation error does not decrease if the noise level is smaller, nor does is increase if the noise level is larger. This can be explained because this method does not include a new local model if the error is too large, but if a region is not covered anymore.

Based on the trace of the MSE of LWPR it looks as if the approximator has not yet converged to its final approximation. Training with 10^6 samples corrupted by the smallest noise variance, gave an MSE validation error of $1.0 \cdot 10^{-6}$ with 236 parameters. It actually fluctuation slightly. The same MSE was found after 1 000 iterations by RKSM.

Both methods handle noise in the targets in their own way. This explains the differences found in these experiments. The LWPR adds and removes regions depending on the coverage. The coverage is little influenced by the noise on the targets. However, in RKSM the noise plays an important role in the selection criterion. The influence of the noise, is that for large noise levels the LWPR comes to a better prediction; but with moderate noise levels, the RKSM comes much faster to good approximations.

4.5 Review

In this chapter, a recursive version of the key sample machine has been developed. This function approximator is called Recursive Key Sample Machine (RKSM). The RKSM is able of updating its *parameters* on-line, as well as altering its *structure.* Evaluations and a comparison showed that the approximator behaves as required, giving a good approximation with little parameters, thus making it potentially applicable for control.

The Key Sample Machine (KSM) has been made recursive by implementing two ideas:

- Assign a weight to each key sample which indicates that the key sample represents a multitude of data.
- Expand the set of key samples if the parameter of the new key sample is unlikely to be zero.

The calculations necessary to implement these ideas, are recursive. The update for the weights of the key samples due to a new sample is *exact*. This means that there is no difference in the weights calculated based on all the samples, or calculated by updating for each new sample. Removal of a key sample from the set of key samples is also *exact*. However, the addition of a sample to the set of key samples is *not exact.* This update cannot be made exact, because the sample previously supplied would be required, and they are omitted. Therefore, it is assumed that the new key sample is uncorrelated with the current set of key samples. This is however, not true, but the best option available in our opinion.

Evaluation of RKSM showed that it was capable of selecting a correct set of key samples from the continuous data stream. The MSE of the on-line version was nearly equal to the MSE of the off-line version. This has been realised by including an omission scheme that omits key samples that have become superfluous. These results suggest that the omission of superfluous key samples should be included for off-line KSM

too. Because the validation error of the approximators are nearly the same for equal key samples, the handling of high-dimensional input spaces is also nearly the same. The on-line approximation can handle high-dimensional input spaces just as well as the off-line approximator.

Similar to KSM, the approximation is dependent on the quality of the data. If the noise variance is low, the approximation is precise, while for a large noise variance, the approximation is rough. However, if the noise level is underestimated, the approximator starts fitting noise. The ridge regression scheme is not capable of reducing this noisy approximation, but by means of structural regularisation, the number of key samples can be limited, which results in a smooth approximation. A disadvantage of this regularisation scheme is that fast fluctuations can no longer be approximated.

With the use of a forgetting mechanism, time-variant functions do not pose a problem.

In a comparison with Locally Weighted Partial Regression (LWPR), it was found that RKSM converged much faster and gave a good approximation with much fewer parameters. However, LWPR was better capable of handling ambiguous targets. RKSM could get a nearly equal MSE as LWPR for a data set with ambiguous targets, but this was achieved by means of structural regularisation, while LWPR handled double targets inherently.

4.5.1 Concerning the problem definition

Real-time constraints

The accuracy of the approximation increases if more key samples are added. The time for the prediction and the memory requirements can be limited as in the off-line approximation: stop including more key samples if the resources are depleted. After the omission of a key sample, a new one may again be added.

The calculation time for an update might become longer than a sample period. Therefore, the calculations for update should not be done in the real-time loop to avoid problems. For this reason, not all the samples supplied to the approximator might get processed, but because of the endless data stream and the high correlation between consecutive samples, this is no problem. It is unlikely that samples on a particular situation repeatedly are omitted, because the motion performed is not repetitive.

Generalisation ability

The evaluation showed that RKSM is comparable to KSM as regards generalisation. The number of inputs, nor the ability to generalise seem to pose problems. The experiments in the next chapter show whether RKSM is really applicable in control.

Five

Real-time control experiments

THE FUNCTION APPROXIMATOR has been developed to be used in a learning control setting. In this chapter it is tested whether the RKSM is capable of storing information so that the learning controller can decrease the tracking error. The experiments will mainly focus on the influence of the function approximator in the control setting. As in previous work the function approximator was the bottle neck to extend the feedforward controller to more inputs, special attention is given to high-dimensional input spaces.

Before we describe the experiments, the setup of the learning controller is treated. The LFFC scheme introduced in chapter 1 is repeated in short in section 5.1 and is extended to include phase-corrected LFFC (De Kruif and De Vries, 2003b). In section 5.2 the design of the different controllers and filters to be used on the Tripod are treated, as well as the settings for the function approximator. The experiments are described in sections 5.3 and 5.4. At the end of the chapter a review is given.

5.1 Learning Feedforward Control

A learning control scheme is only useful, when the behaviour of the plant is not fully known. If the plant has been identified completely, then this information can be used directly and no learning is required. Because the behaviour of the Tripod is not known accurately, a learning controller is useful. The information that is deduced by the learning controller may be used for future controller design.

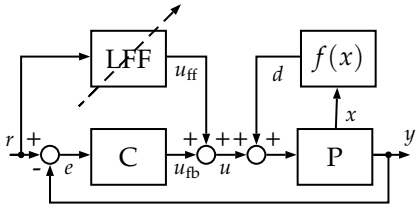


Figure 5.1: Learning FeedForward Control scheme

5.1.1 Traditional Learning Feedforward Control

The control scheme that is used to test the function approximator is Learning FeedForward Control (LFFC) (De Vries et al., 2001; Otten et al., 1997; Starrenburg et al., 1996; Velthuis, 2000; Velthuis et al., 1998). How this method uses an approximation of the state-dependent effects to compensate for them, was treated in chapter 1. The block diagram of the LFF control scheme is repeated in figure 5.1. After learning, an approximation of f is present in the block LFF. The inputs to the approximation are the states commanded (r) instead of the true states (x). In order for the compensation to be effective, the output of the approximator with x used as input, should be close to the output with r as input. This means in practice, that the state-dependent effects should be smooth and that the controlled system should follow the states commanded well.

However, before the LFF can make a prediction to compensate for the state-dependent effects, it has to form an approximation. In order to learn the relation $f(x)$, the function approximator needs samples that represent it. Direct approximation of $f(x)$ is not possible because the states x, as well as the disturbance signal d, are generally not available. Similar to the situation for prediction, the states commanded are used instead of the true states to learn this relation.

In the literature on LFFC, the feedback signal u_{fb} is used as estimate of the disturbance signal d and is therefore used as target for the learning mechanism. The transfer function from the signal d to the feedback signal u_{fb} is calculated, omitting the arguments for notational convenience, as:

$$\frac{u_{\text{fb}}}{d} = -\frac{CP}{1+CP}. \tag{5.1}$$

In this derivation it is assumed that the disturbance signal is decoupled from the plant, which is allowed if $f(x)$ is smooth enough. When $CP \gg 1$, which is generally the case in the lower-frequency range, this transfer function is nearly one. However, for higher frequencies, the transfer

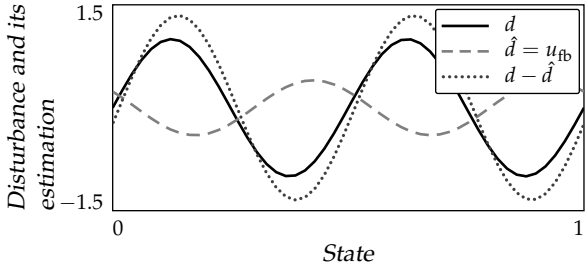


Figure 5.2: Instability due to phase shift

function is smaller than one and there is a phase lag, assuming that the plant and the controller are proper.

As shown in Velthuis (2000), a phase lag diminishes the performance of the LFFC and can even lead to instability. If the phase shift of the transfer function (5.1) becomes larger than 90°, and the feedback signal, u_{fb} is used as approximation of the signal d, then the estimate for compensating the disturbance does not decrease the influence of the state-dependent effects, but it will increase the influence. Refer to figure 5.2. In this figure a path is travelled with a constant velocity and the disturbance signal as well as the estimate of it are recorded as functions of the position. The LFFC controller learns the estimate $\hat{d}$ as a function of the position. If this estimate is used to counteract the state-dependent effect, the difference between the state-dependent disturbance and its estimate is bigger than the original disturbance signal, resulting in growth of the tracking error. This undesirable growth is shown by the dotted line in figure 5.2.

5.1.2 Phase-corrected Learning Feedforward Control

As the phase shift causes the tracking error to increase, an other estimate of the the disturbance signal d was sought for (De Kruif and De Vries, 2003b). Instead of using the feedback signal, a filtered error signal was used. This is similar to what is done in ILC (Arimoto et al., 1984; Longman, 1998, 2000; Moore, 1992). The transfer function from the disturbance signal to the error signal is given as:

$$U = \frac{e}{d} = -\frac{P}{1 + CP}. \tag{5.2}$$

The inverse of this transfer function gives the disturbance signal, based on the error signal. The inverse of U is referred to as the learning filter

and is given as:

$$L = U^{-1} = \frac{d}{e}. \tag{5.3}$$

However, before this learning filter can be used to obtain a disturbance estimate, the following should be noted:

1) If $U(s)$ contains zeros in the right half plane and is causal, then its inverse has poles in the right half plane and is therefore unstable.

2) Since both $P(s)$ and $C(s)$ are usually proper, $U(s)$ is proper. Therefore, $L(s)$ is not proper, resulting in amplification of high frequencies.

3) $U(s)$ is uncertain, especially for higher frequencies, which makes $L(s)$ uncertain.

The first of these items can be coped with by the Zero Phase Error Tracking Control (ZPETC) method of Tomizuka (1987). This method gives a stable inverse of a discrete plant, and compensates for the phase shift that is caused by the uncancelled zeros. The result of the ZPETC algorithm on a proper plant is a transfer function with a negative delay, i.e. it is anti-causal and future samples are needed for its estimation. In an on-line control setting there is no future information. Hence, it is not feasible to apply this learning filter to learn at the *current commanded state.* However, instead of trying to update the approximator for the current commanded state, it *is* possible to update the feedforward controller for the *commanded states several time steps in the past,* because for these commanded states several steps into the future *are* known. Delaying the update makes the filter causal.

The phase-corrected LFFC scheme is shown in figure 5.3. In this figure the solid lines denote the control signals, the dashed lines the learning signals. The learning signals are delayed so that the anti-causal learning filter L can be implemented; the control signals are obviously *not* delayed. Signals without argument are signals as a function of the time t.

In the figure two low-pass filters Q are included. The first filter, Q_1, is included to counteract the amplification of noise at high frequencies and to deal with the uncertainties of the plant model. When the model of the plant does not agree with the true plant's behaviour, which is likely to happen for higher frequencies, the disturbance estimate is wrong. This incorrect information should not be incorporated into the approximation and therefore it is suppressed by the use of a filter. This filter counteracts the problems described in points 2 and 3 in the list given above.

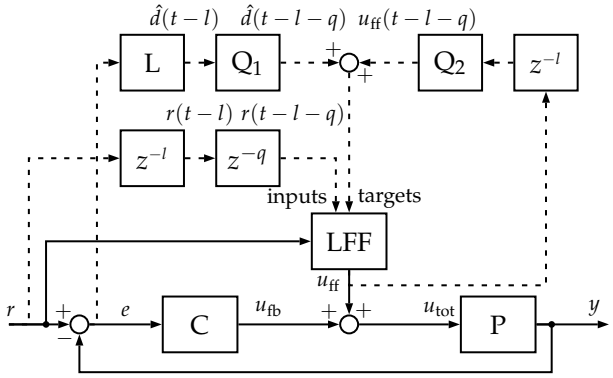


Figure 5.3: The phase corrected LFFC scheme

The second low-pass filter, Q_2, is merely a safety measure. The disturbance estimate gives the estimated disturbance that is still acting on the plant. The true disturbance signal is already partially compensated for by the feedforward controller and therefore the output of the function approximator should be added to the estimated disturbance. Q_2 is included to counteract possible glitches. The bandwidth of this filter can be much larger than the bandwidth of filter Q_1.

In figure 5.3 the delays of the reference states are also indicated. z^{-1} is slightly abused to denote the delay operator. These delays cause the delays of the learning filter, l, and the low-pass filter, q, also to be perceived by the state. Therefore, the disturbance estimate and the states commanded are equally delayed and can be used to learn the state-dependent effects.

5.1.3 Experiment

In De Kruif and De Vries (2003b) the phase-corrected LFFC scheme is experimentally compared with the traditional LFFC. Only the result of this comparison is given in this thesis to show the difference in performance of these learning schemes. For more information on the experiment, we refer to the paper.

The experiments were done with a single linear motor, which was not connected to additional structures as are the linear motors of the Tripod. The dominant effects in a linear motor are:

Cogging: This effect originates from the attraction between the permanent magnets located in the stator and the coil's iron core located in the translator. The attraction force depends on the position of

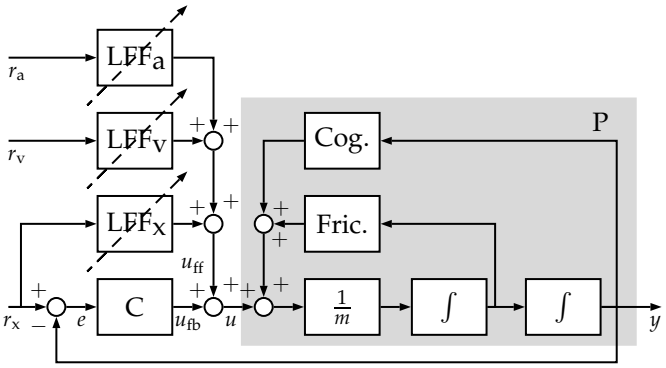


Figure 5.4: Control setting for one linear motor with additive state-dependent effects; the control setting exploits additive property by using three one-dimensional feedforward controllers (parsimonious LFFC)

the translator and to counteract this force, a position-dependent compensation-force should be learnt. The position is therefore required as input.

Friction: Friction mainly depends on the velocity of the translator relative to the stator. However, due to possible curvature of the linear guides, the friction also depends, to some extent, on the position. The friction force can be predicted largely if the velocity is known and can thus be compensated for.

Unknown mass: The mass relates the acceleration and the force.

The state-dependent effects were assumed to be additive and each depend on only *one* state, see figure 5.4. With this assumption, the effect of cogging and friction can be compensated for by a set of independent learning feedforward controllers with only one input. This setup is known as a parsimonious learning feedforward controller (De Vries et al., 2001) and illustrated in figure 5.4. As the function approximators only had to approximate a relation based on one input, a BSN was used.

The tracking error at the end of the motion is given in figure 5.5. In this figure, the dashed gray line gives the tracking error for the traditional LFFC and the solid black line for the phase-corrected LFFC. Based on this result, it can be seen that the fast oscillations, due to the cogging, are much better compensated for by phase-corrected LFFC. However, a slowly fluctuation error remains. This error is due to position dependent friction. The linear guides are not exactly straight, so, apart from the velocity, the friction depends on the position. This effect cannot be

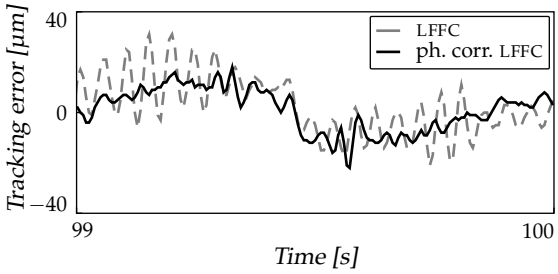


Figure 5.5: Tracking error of LFFC and phase corrected LFFC

counteracted if only effects depending on one state are compensated for. The use of a learning feedforward controller that compensates for higher-dimensional effects is considered in the remainder of the chapter.

5.2 Implementation for the Tripod

There are several state-dependent effects in the Tripod that can be compensated for by the learning mechanism. Which inputs are required to compensate for these effects is first investigated. After this, the feedback controller and the filters are designed.

5.2.1 Inputs to feedforward controller

Only those effects can be compensated for, which the feedforward controller can predict by its inputs. Effects that are dominantly present in the Tripod are the dynamic coupling between the motors and the motor characteristics treated in section 5.1.3. The inputs required for the motor characteristics are treated before. However, the mass that one specific motor has to move, depends, for the Tripod, on the location of all the motors. By incorporating both the acceleration in the feedforward controller and the positions of the other motors, the force required to accelerate the motor can be learnt

In order to compensate for the coupling between the motors, the feedforward controller needs to know the acceleration as well as the position of the other moving motors.

Which inputs were actually used depended on the experiment that was carried out. Experiments were carried out with a varying number of motors moving. Commanded states of motors that are not moving, are not necessary as input. Furthermore, some inputs may not have a considerable predictive power and could be omitted without giving

a significant error increment. Different possibilities were tried and for each experiment the used inputs are given.

5.2.2 Feedback controller

The state-dependent effects are assumed to be compensated for by the learning element. The goal of the feedback controller is therefore not to guarantee performance, but to create sufficient reference-following capabilities for the commanded states to be used for learning and compensating for the state-dependent effects. A SISO feedback controller is developed for each motor. These feedback controllers should be robust enough to cope with the state-dependent effects which are considered disturbances. For the design of the feedback controller, the behaviour of each motor is modelled as a moving mass:

$$P(s) = \frac{1}{ms^2}. \tag{5.4}$$

The ZPETC algorithm limits the possible feedback controllers for this plant. This algorithm forms the learning filter by calculating a stable inverse of $U(s)$. However, if the transfer function $U(s)$ contains poles in the origin, the inverse is undetermined. This means that the controller cannot contain an integrator. A PD-controller is therefore chosen for the control of each motor.

The transfer function of the controller is given as:

$$C(s) = K_{\mathrm{p}} \frac{s\tau_{\mathrm{d}} + 1}{s\beta\tau_{\mathrm{d}} + 1}. \tag{5.5}$$

Based on the (estimated) mass of the motor together with a given bandwidth, the τ_{d} and the K_{p} are calculated such that the maximal phase shift occurs at the 0 dB crossing of the modulus of the loop transfer function, and that this crossing is at the specified bandwidth. The bandwidth of the controlled system was increased, until it almost started oscillating. The resulting closed loop bandwidth of the controlled system is 46 [Hz]. The peaks of the complementary sensitivity and the sensitivity are at 17 and 25 [Hz] respectively. The RMS of the tracking error for the PD-controller is about 175 [µm] for a motion similar to the motions that were commanded in the experiments. The values for the parameters were $K_{\mathrm{p}} = 8.09 \cdot 10^4, \tau_{\mathrm{d}} = 8.43 \cdot 10^{-3}$. β was selected to be 0.1. A bode plot of the sensitivity function and the complementary sensitivity function are given in figure 5.6.

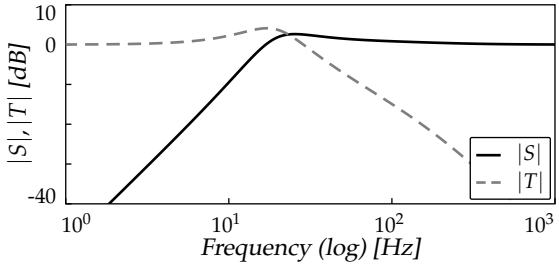


Figure 5.6: Bode plot of (complementary) sensitivity function

5.2.3 Learning filters

To determine the learning filter L, the plant as well as the controller have to be discretised. The discretisation of the plant was done in such a way that the output values of the discretised model were identical to those of the continuous model at the sampling moments (Åström and Wittenmark, 1997):

$$P(z^{-1}) = (1-z^{-1})\mathcal{Z}\mathcal{L}_{\mathrm{p}}^{-1}\left(\frac{P(s)}{s}\right) \tag{5.6a}$$

$$= (1-z^{-1})\left(\frac{T^2 z^{-1}(1+z^{-1})}{2m(1-z^{-1})^3}\right) \tag{5.6b}$$

$$= \frac{T^2 z^{-1}(1+z^{-1})}{2m(1-z^{-1})^2}. \tag{5.6c}$$

In this, T stands for the sampling time, $\mathcal{L}_{\mathrm{p}}^{-1}$ for the inverse Laplace transform, $\mathcal{Z}$ for the z-transform and m for the mass. The PD-controller was approximated in discrete time by using the Matched Pole Zero method of Franklin, Powel, and Emami-Naeini (1994). This results in:

$$C(z^{-1}) = K_{\mathrm{pd}}\frac{1-\alpha z^{-1}}{1-\gamma z^{-1}} \tag{5.7}$$

with

$$K_{\mathrm{pd}} = K_{\mathrm{p}}\frac{1-\gamma}{1-\alpha}, \quad \alpha = e^{-T/\tau_{\mathrm{d}}} \quad \text{and} \quad \gamma = e^{-T/\beta\tau_{\mathrm{d}}}. \tag{5.8}$$

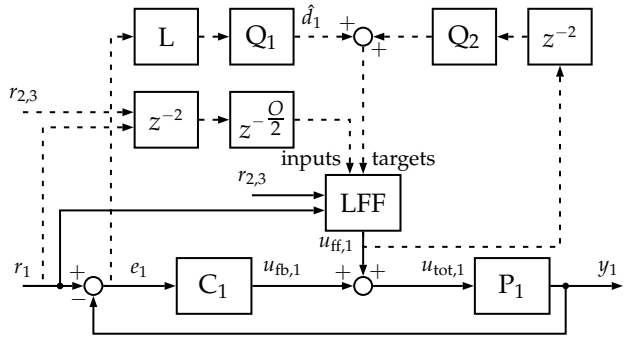


Figure 5.7: Learning control scheme for motor 1 of the Tripod

Applying the ZPETC method to the discretised function $U(z^{-1})$ (5.2) results in the learning filter $L(z^{-1})$ (Van Dijk, Tinsel, and Schrijver, 2000):

$$\frac{\hat{d}}{e} = L(z^{-1}) = \frac{\Big(2m(1-z^{-1})(1-\gamma z^{-1}) + 2K_{pd}z^{-1}(1+z^{-1})(1-\alpha z^{-1})\Big)(1+z^{-1})}{4T^2(1-\gamma z^{-1})} z^2. \tag{5.9}$$

The z^2 at the end of the equation shows that the errors two time steps in the *future* are needed to calculate the disturbance estimate *now.* In order to implement this filter, the update of the function approximator has to be delayed by two time steps. The delays of the signals are included in figure 5.7, in which the control scheme for one of the motors is depicted. In this figure the r_i represents the reference signals of motor number i, z^{-2} denotes the delay due to the learning filter. The second delay, $z^{-O/2}$, is to compensate for the delay introduced by the low-pass filter.

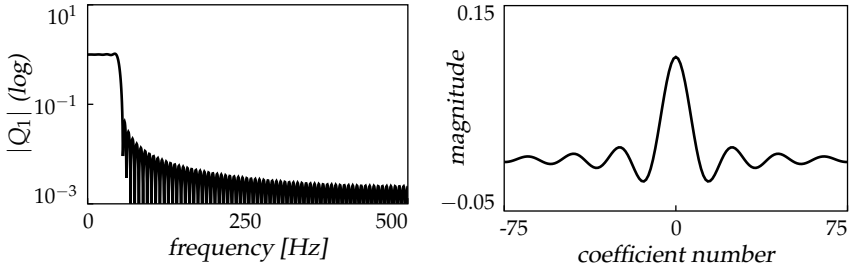
The low-pass filters Q_1 and Q_2 are implemented as FIR-filters:

$$y(k) = b(0)x(k) + b(1)x(k-1) + \ldots + b(O)x(k-O). \tag{5.10}$$

The coefficients b have been calculated to implement a low-pass filter with an optimal frequency response in a least squares sense (Schlichthärle, 2000). This results in the following coefficients:

$$b\Big(i + \tfrac{O}{2}\Big) = \frac{\omega_c \sin(\pi\omega_c i)}{\pi\omega_c i}, \quad \text{for} \quad i = -\tfrac{O}{2}, \ldots, \tfrac{O}{2}, \tag{5.11}$$

in which i is the coefficient number, O the order of the filter and ω_c the normalised cutoff frequency. The order of the FIR-filter has to be even



(a) Frequency response (b) Filter coefficients

Figure 5.8: 150$^{\text{th}}$-order low-pass FIR filter Q_1 with $\omega_c = 50$ [Hz]

for this design. The frequency response of this filter for a 150$^{\text{th}}$-order filter (with 151 coefficients) as well as the magnitude of its coefficients are given in figure 5.8. In the control setup the cut-off frequency of Q_1 was $\omega_{c,1} = 50$ [Hz], for Q_2 $\omega_{c,2} = 300$ [Hz] and the order for both filters was 150.

A FIR filter was used because of its exact linear-phase frequency response (Schlichthärle, 2000). A linear-phase frequency response means that the filter implements a pure delay. Consequently, the inputs to the feedforward controller can be delayed with the same amount, as shown in figure 5.7. As both the target and the input of the function approximator are equally delayed, the function approximator learns the correct relation. The order of filter Q_1 and Q_2 should be the same, otherwise the feedforward signal is not added to the disturbance estimate at the correct time. Figure 5.7 shows the control scheme for one motor.

5.2.4 Function approximator settings

With the control scheme fully specified, some settings of the function approximator have to be decided on. The same settings are used for the on- and off-line function approximators, where applicable.

Set of functions

First, the set of functions with which the learning mechanism has to approximate the data is selected. This is done by choosing a dual indicator function. As the relation that is sought is unknown, an indicator function that can approximate a broad set of functions should be used. Therefore, splines are used for the approximation. The order of

the spline is selected to be one, so that a piecewise linear approximation is found. The advantage of this is that the calculation time is little. This dual indicator function for a one-dimensional input is given, see example 3.3, as:

$$\tilde{f}_i(x) = 1 + \min(x, x_i). \tag{5.12}$$

This dual function will result in a non-smooth approximation. The dual indicator function for a second order spline, which is smoother, is (Vapnik, 2000):

$$\tilde{f}_i(x) = 1 + xx_i + \tfrac{1}{2}xx_i \min(x, x_i) - \tfrac{1}{6}\min(x, x_i)^3. \tag{5.13}$$

To restrict the calculation time, the first order spline is used. Instead of splines, B-splines could be used to approximate the data. This option is not used because of their local support, the extra calculation time and an equivalent set of functions the B-splines describe. The same holds for the indicator functions that are given in Girosi, Jones, and Poggio (1993) which result in piecewise linear functions. Although the set of functions they can describe, are equal, transient behaviour can be different.

In order to construct to a multi-dimensional kernel of (5.12), the product of the uni-dimensional kernels can be used (Vapnik, 2000):

$$\tilde{f}_i(\mathbf{x}) = \prod_{j=1}^{n_{\text{dim}}} f_i(x^j). \tag{5.14}$$

The corresponding set of functions is the tensor product of the co-ordinatewise basis functions (Vapnik, 2000, Theorem 6.1). This set of functions can approximate a general function of the form:

$$\hat{y}(\mathbf{x}) = g(x^1, x^2, \ldots). \tag{5.15}$$

If knowledge is available that the data can be approximated by an additive form:

$$\hat{y}(\mathbf{x}) = g_1(x^1) + g_2(x^2) + \ldots \tag{5.16}$$

Then, instead of the product in (5.14), a sum can be used:

$$\tilde{f}_i(\mathbf{x}) = \sum_{j=1}^{n_{\text{dim}}} f_i(x^j). \tag{5.17}$$

This yields a smaller set of functions for the approximation. If the data can be approximated by this smaller set, a better approximation is found.

Combinations of (5.14) and (5.17) are possible (Cristianini and Shawe-Taylor, 2000). Furthermore, also an ANOVA decomposition may be made by the dual indicator functions (Stitson, Gammerman, Vapnik, Vovk, Watkins, and Weston, 1997).

Because we assume to be ignorant of the relation underlying the data, we use a kernel of the form (5.14) in our RKSM scheme. If the goal of the experiments was solely to decrease the tracking error, different kernels should be tried to see which one fits the data best.

Significance level of hypothesis testing

The significance level for rejecting the hypothesis by which a key sample is included (3.53) and (4.55), is set at some fixed value. The actual value of this significance is of no influence, because the noise estimate can be used to alter the inclusion, see section 3.3.4 and 4.2.3. The noise estimate is used as a design parameter and tuned throughout the experiments.

The omission scheme is not used, due to problems with the switching of the stacks at the interrupts in the real-time operating system. When the omission scheme is not used, the set of key samples will contain superfluous key samples. However, as seen in figure 4.5, the approximation remains good. The disadvantage is that the number of key samples is greater than necessary and hence the computation effort is (significantly) greater than needed.

Forgetting factor

The forgetting factor for the on-line approximation allows for a time variant function to be approximated. The state-dependent effects of our setup are not regarded time variant.

Although a small forgetting factor also decreases the possibility of glitches as given in section 4.2.4, the multiplication of the weight matrix with a constant does take calculation time. Therefore, in the experiments, the forgetting mechanism is not used. If it is found that glitches pose a problem to the performance, measures should be taken.

Structural regularisation

Structural regularisation is used in our experiments. Because we want to allow for fast changes in the approximation, the value of the minimal residual of the new information is set low. It is in the order of 10^{-10}, which is 10^5 times larger than the numerical precision of the computer.

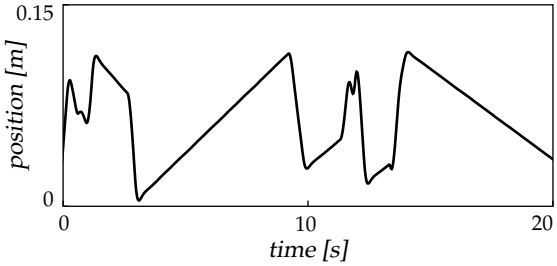


Figure 5.9: First twenty seconds of a motor path

Slow start

In the beginning of the operation the setup can display some transient behaviour, e.g. when the first reference sample is different from the first measurement. The function approximator should not learn these transients. Therefore, in the off-line experiment, the first seconds are removed from the data set.

In the on-line experiments, the learning feedforward controller starts slightly later with the approximation of the data. Furthermore, the feedforward signal, i.e. the prediction of the function approximator, is multiplied with a scaling factor. This factor starts at zero and increases smoothly to one in six seconds. The feedforward signal is smoothly activated to avoid jumps and other stepwise excitations.

5.3 Off-line results

The off-line function approximator is tested first to see if the KSM can be used in learning control. Because we are mainly interested in the on-line applications, we only investigate the situations in which one motor is allowed to move in the off-line experiment.

5.3.1 Motion profile

The motion that has been used for the training and for the evaluation, was a concatenation of third order motions. This path was randomly generated only once, so that evaluations can be performed on these pseudo-random paths. The sequence contained no repetitions of the commanded motions. The position commanded of motor number one for the first twenty seconds is shown in figure 5.9. The stroke of the

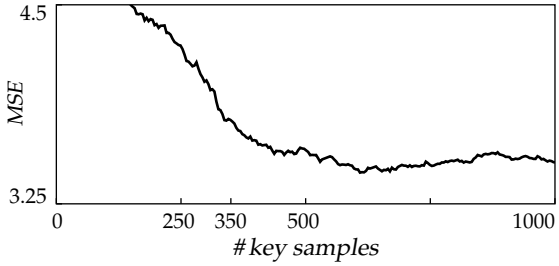


Figure 5.10: Validation error for off-line approximation

motor was set at 0.125 [m], which keeps the platform within its safe operation region. The maximum speed was 0.3 [m/s] with a maximum acceleration of about 4 [m/s^2].

5.3.2 Approximate

The disturbance estimate and the delayed states obtained by the scheme given in figure 5.7 were used to approximate the state-dependent effects. The function approximator first collected all the samples, and after the training run, the relation was learnt.

The training run contained 200 000 samples. Out of these two data sets were made: a training set of 7 500 samples and a validation set of 7 500 samples. The training set was used by the KSM to find a set of key samples, while simultaneously the MSE on the validation set was calculated. The KSM includes one key sample a time, and therefore a whole set of approximators was made with different numbers of key samples. The MSE of these approximators on the validation set is given in figure 5.10. Above approximately 350 key samples, the MSE does not significantly decrease. Therefore, an approximator with 350 key samples was used. The approximation was found in approximately 240 [s] on a Pentium IV 2.4 GHz.

5.3.3 Evaluation

The approximation found was applied to the evaluation motion. The last ten seconds are shown in figure 5.11. The RMS before learning was 183 [µm], while after learning, it was reduced to 22 [µm]. Based on this observation, it may be concluded that the off-line function approximation is capable of learning a relation in a learning control setting with 3 inputs. With the approximation found by KSM, it is possible to reduce

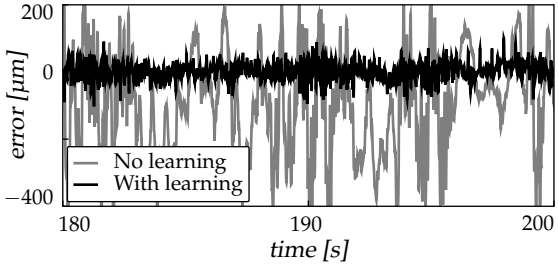


Figure 5.11: Tracking error with off-line learning, only one motor is moving

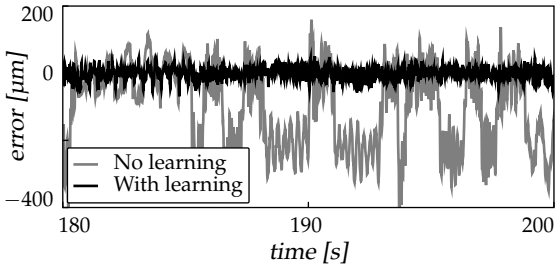


Figure 5.12: One motor moving; tracking error with and without learning

the tracking error significantly. For thorough investigation of the residual after learning, see section 5.4.1.

5.4 On-line results

In the experiments in which the approximator has to find the relation on-line, each of the motors was supplied with its own reference motion. Similar to the off-line experiment, these paths were a concatenation of third order motions. The stroke, maximum velocity and maximum acceleration were the same as before.

5.4.1 One moving motor

The first set of experiments was carried out with only one motor moving. Because there is no dynamic coupling if the other motors are stationary, only the position, velocity and acceleration of the moving motor were fed to feedforward controller. The learning mechanism only had to learn the characteristics of the motor.

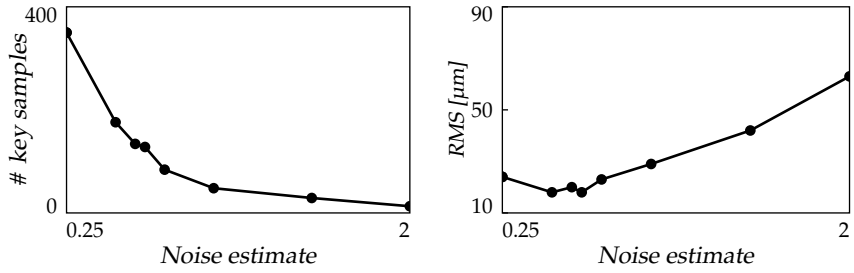
The tracking error of the motion with learning and without learning is given in figure 5.12. The last twenty seconds of the experiment show the improvement due to learning. The RMS of the tracking error in the second half of the experiment without learning is 170 [µm], while after learning the RMS has decreased to 18 [µm]. It can be concluded that the tracking error has decreased significantly, due to the phase-corrected learning feedforward control with a RKSM learning mechanism. The learning mechanism is capable of finding a relation in the data stream with three inputs by which the effects of the motor characteristics are compensated for.

The compensation of the cogging force can be observed at e.g. $t \approx 192$ [s]. Before the learning, a periodic tracking error is seen here, while after the training, this large tracking error does not show anymore. The effect of friction has also diminished greatly. The effect of the friction before learning can be seen by the great errors with a non-zero mean. If the direction of motion is turned around, a jump in this error is seen, e.g. at $t \approx 190$ [s]. After learning, this effect disappears. A sudden acceleration at e.g. $t \approx 194$ [s] gave a large tracking error before learning because the required force to come to this acceleration was not fed forward. After learning, this great error was compensated for.

The error found after learning is smaller than in the off-line experiment, 18 against 22 [µm]. This difference can be explained because in the on-line experiment the approximation was used to improve the tracking behaviour during the experiment. Therefore the commanded states were closer to the true states, and the approximation based on the commanded states approximated the state-dependent effects better. Furthermore, more data is processed by the on-line function approximator. It should be noted that the error *before* the learning was also slightly larger in the off-line experiment, 183 against 170 [µm].

Influence of noise estimate

The effect of the noise estimate on the number of key samples as well as on the tracking error is shown in figure 5.13. As expected, the number of found key samples grows when the noise estimate is smaller. In the corresponding tracking error, the RMS decreases until a certain level and if the noise estimate is reduced further, it starts rising again. There are two explanations for this behaviour. First, because the number of key samples grows, the number of training samples for each key sample becomes less, and therefore, the target of the key sample is less certain. This results in a larger variance as regards the prediction, so that



(a) Number of key samples (b) RMS of tracking error

Figure 5.13: Influence of the noise estimate

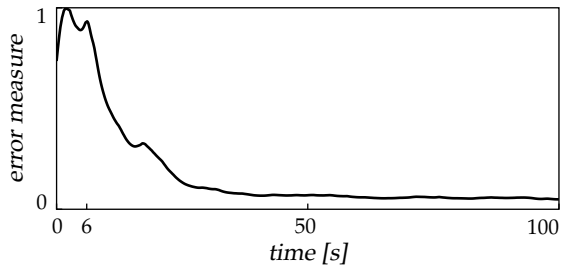
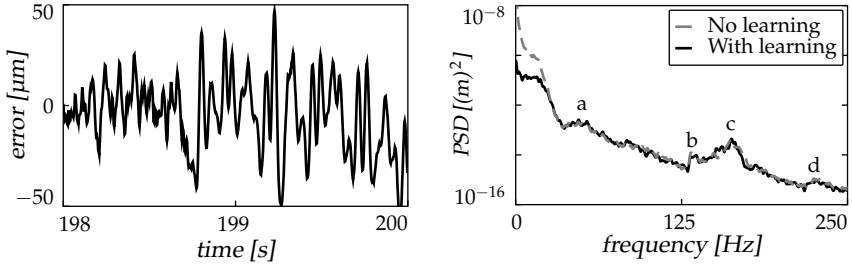


Figure 5.14: Determination of convergence speed. The error measure is the low-pass filtered squared error

the state-dependent effects cannot be compensated for equally well anymore. Second, the noise or residual vibrations are fitted. If the noise estimate becomes too small, random phenomena and residual vibrations are fitted that do not contribute to a better prediction. These two factors limit the precision with which the function can be approximated.

Convergence speed

In order to investigate the convergence speed, the squared error has been filtered by a low-pass filter. The result is normalised between zero and one and depicted in figure 5.14. Due to the slow start, the feedforward signal is fully active only after six seconds. This graph shows that after approximately 25 seconds of learning the error does not decrease significantly anymore. This means that with 25 000 samples — which are closely correlated because they describe a path — an approximation of the three dimensional state-dependent disturbances was found; structure as well as parameters.



(a) Residual after learning (b) PSD of residual

Figure 5.15: The residual

Verification

Although the learning feedforward controller significantly reduced the tracking error, there still was a residual. A zoom of this residual is given in figure 5.15(a). Vibrations can be seen in the peaks of the Power Spectral Density (PSD) plotted in figure 5.15(b). In this figure 'a' is located at a peak between 40 and 55 [Hz], 'b' between 130 and 135 [Hz], 'c' at a peak between 160 and 170 [Hz] and 'd' at a peak between 220 and 230 [Hz].

In the PSD plot, it can be seen that the power in the lower frequencies of the signal has reduced significantly. The power in the frequency range from 0 to 20 [Hz] is a factor 10 to 1 000 smaller after learning. Above approximately 25 [Hz], which happens to be the frequency where the sensitivity function peaks, the power spectrum for the learnt and the unlearnt method is equal.

Some of the vibrational modes that were found by exciting the plant in rest, summarised in table 5.1, are again found in the PSD-plot. The first peak, denoted: 'a,' can be related to the vibrations around the horizontal axes and the horizontal translational mode which are all about

Table 5.1: Characteristics of the Tripod

Vibrational modes			**Motor characteristics**		
Vert. axis	65	[Hz]	Cogging amplitude (pp)	30	[N]
Hor. axes	40	[Hz]	Cogging period	24	[mm]
Horizontal	40, 145	[Hz]	Coulomb friction	12	[N]
Vertical	225	[Hz]	Mass	$5.8 - 6.2$	[kg]
			Dynamic coupling	-1.1	[$\mathrm{N/m\,s^{-2}}$]

40 [Hz]. The vertical vibration mode at circa 225 [Hz] is found as peak 'd.' The vibration around the vertical axis of 65 [Hz] is not present in the residual because it is hardly excited by the actuators. One of the peaks 'b' or 'c' is likely to be related to the other vertical mode. Because these vibrations cannot be predicted with the states commanded used, the LFFC-scheme cannot compensate for them. Therefore, the amplitude of these peaks remains the same.

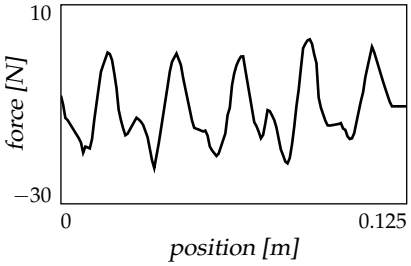
The amplitudes of these vibrations form a lower bound for the noise estimate, therefore, they determine the maximum achievable performance. If the noise estimate is estimated too small, then each top of these vibrations is included as key sample, resulting in too many, incorrect, key samples. This further clarifies figure 5.13.

The power in the frequency range after approximately 25 [Hz] is not altered by the learning control scheme. This 25 [Hz] is close to the peaking of the (complementary) sensitivity function. Experiments in which the bandwidth of the controller and that of the low-pass filters were changed, showed that these had influence on the PSD of the tracking error. It is recommended that in future research it is investigated *why* the tracking error does not decrease for frequencies larger than 25 [Hz]. However, due to limited time it could not be investigated further in this research.

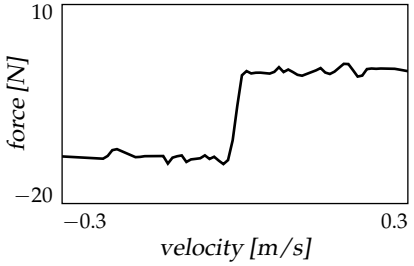
The contents of the function approximator is shown in figure 5.16. In this figure only one input at a time is altered, and therefore effects that depend on several states are not noticed. The mass is found to be 5.8 [kg] by fitting a straight line in 5.16(c). The effects shown in figure 5.16 correspond to the magnitude of the effects that were found in a set of preliminary experiments. The motor characteristics that were extracted form these experiments are summarised in table 5.1. These numbers were obtained by the traditional LFFC scheme, with motion profiles especially designed. For example, to measure the cogging force, a very slow motion is made; to measure the friction force, a wide variety of velocities is presented and to measure the mass, a wide variety of accelerations is commanded.

5.4.2 Two motors moving

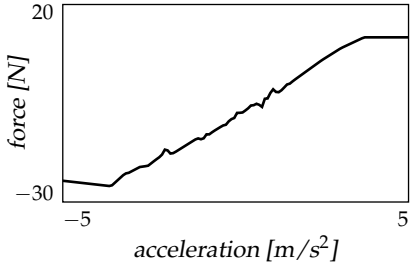
In the next set of experiments two motors are moving. We are only interested in the tracking error of motor number one. Motor number two is merely used to act as a disturbance. The control scheme for motor number one contains the phase-corrected learning feedforward controller as before. Motor number two however, is controlled (only) by a



(a) Identified cogging, $\hat{f}(x, 0, 0)$



(b) Identified friction, $\hat{f}(0, \dot{x}, 0)$



(c) Identified mass, $\hat{f}(0, 0, \ddot{x})$

Figure 5.16: Identified state dependent effects

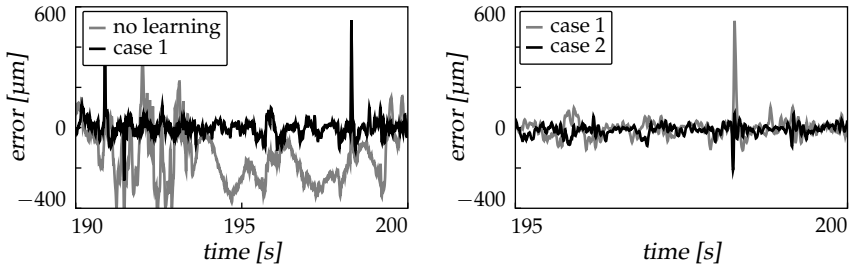
PD-controller.

Which inputs are of importance to construct the feedforward controller is less obvious. The extra effect that is to be compensated for, is the influence of the second motor. Although the dynamic coupling does not exert big forces on motor number one, for big accelerations the forces *are* clearly observable. Different inputs are considered for the learning mechanism in this set of experiments. Three cases are considered:

Case 1) Only the position, velocity and acceleration of motor number one are used as inputs. The dynamic coupling is disregarded. So, $\mathbf{x} = (r_{\mathrm{x},1}, r_{\mathrm{v},1}, r_{\mathrm{a},1})$.

Case 2) Apart from the previous inputs, the acceleration of motor number two is added as input. This acceleration can largely predict the coupling force from motor number two on motor number one: $\mathbf{x} = (r_{\mathrm{x},1}, r_{\mathrm{v},1}, r_{\mathrm{a},1}, r_{\mathrm{a},2})$.

Case 3) The commanded states of the moving motors are used. This means that the position, velocity and acceleration of both motors are used as inputs. With this set of inputs, all the effects should



(a) No learning and learning with the commanded states of one motor as input

(b) learning with and without the addition of the acceleration of motor number two

Figure 5.17: Two motors moving; tracking error for different inputs

be compensated for. However, some inputs might have a marginal influence on the output of motor number one. The inputs are: $\mathbf{x} = (r_{x,1}, r_{v,1}, r_{a,1}, r_{x,2}, r_{v,2}, r_{a,2})$.

In figure 5.17(a) the tracking error, for the first case, of motor number one is shown before and after learning. The tracking error is comparable with the tracking error graph for only one moving motor. However, significant peaks are found at several instances, e.g. at $t \approx 198$ [s]. This peak is due to an acceleration of motor number two. Because the acceleration of motor number two is not an input in this case, the learning mechanism has no means of prediction. Therefore, the peak remains after learning.

In case 2, which adds the commanded acceleration of motor number two as input to the feedforward controller of motor number one, this error peak should lower. In figure 5.17(b) the tracking error of the last five seconds is shown with and without this fourth input. It can be seen clearly that the peak due to dynamic coupling is greatly diminished by adding this input.

The initial high peak in the error, due to the dynamic coupling, is compensated for by adding the acceleration of motor number two. However, residual vibrations *remain* after learning. This can be explained by using a simple model of the plant, refer to figure 5.18. In this model, the linear motor is denoted as m_1; F is the force acting directly on it; m_2 represents the platform and F_{dist} is the force applied, due to the acceleration of motor number two on the platform. The spring represents the compliances in the system. LFFC calculates a force, F, based on the states. In this figure, the F_{dist} can be related to the acceleration of motor

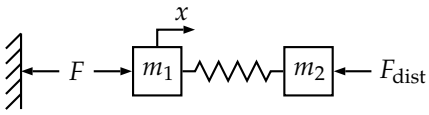


Figure 5.18: F can be used to compensate for effect of F_{dist}, but not for the vibrations due to the excitement of the system

two, and for argument's sake, we assume that it is known.

LFFC calculates a force to compensate for the effect of F_{dist} *based only on* F_{dist}: $F = f(F_{\text{dist}})$. This compensation force can compensate for the initial, big error at the position x, but this disturbing force will also excite the system and introduce vibrations. Because the compensation force does not incorporate the dynamics, it cannot compensate for these vibrations. This would only be possible if the dynamics were included: $F = f(F_{\text{dist}}, t)$. This behaviour can also be seen at the Tripod; besides the direct effect on the position of motor one, the acceleration of motor two excites the system. The direct effect can be compensated for in the current setting, but the resulting vibrations cannot.

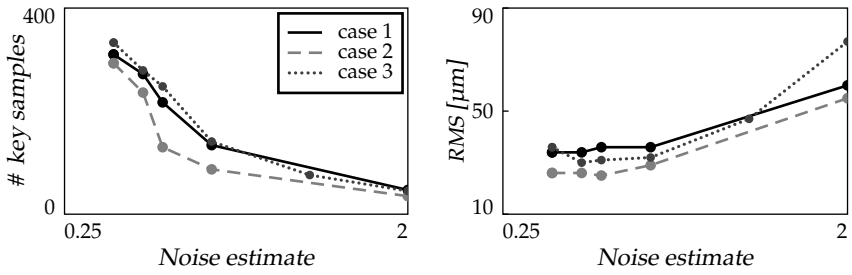
The third case uses the commanded states of both motors as inputs to the learning controller for motor number one: $\mathbf{x} = (r_{\text{x},1}, r_{\text{v},1}, r_{\text{a},1}, r_{\text{x},2}, r_{\text{v},2}, r_{\text{a},2})$. The learning mechanism with these six inputs was capable of compensating for the effects due to the coupling between the motors. The results of the experiments with the different inputs, are given in figure 5.19. The number of key samples and the RMS of the tracking error are plotted as function of the noise estimate. The tracking error without the learning component is 182 [µm]. It can be seen that adding the commanded position and velocity of motor number two as input, i.e. going from four to six inputs, does not further decrease the error.

Based on figure 5.19, we may conclude that the commanded states of motor number one *and* the commanded acceleration of motor number two as input to the learning controller, give the best performance. Relating this to the learning setting of figure 2.1(a):

$$\mathbf{x} = (r_{\text{x},1}, r_{\text{v},1}, r_{\text{a},1}, r_{\text{a},2}) \quad \text{and} \quad \mathbf{x}_{\text{u}} = (r_{\text{x}_2}, r_{\text{v}_2}). \tag{5.18}$$

The tracking error as well as the number of key samples is the smallest for all the noise estimates if these four inputs are used. The best achieved RMS of the tracing error is 25 [µm] for a noise estimate of 0.75 [N]. For smaller noise estimates, the tracking error does not increase significantly, but the number of key samples does.

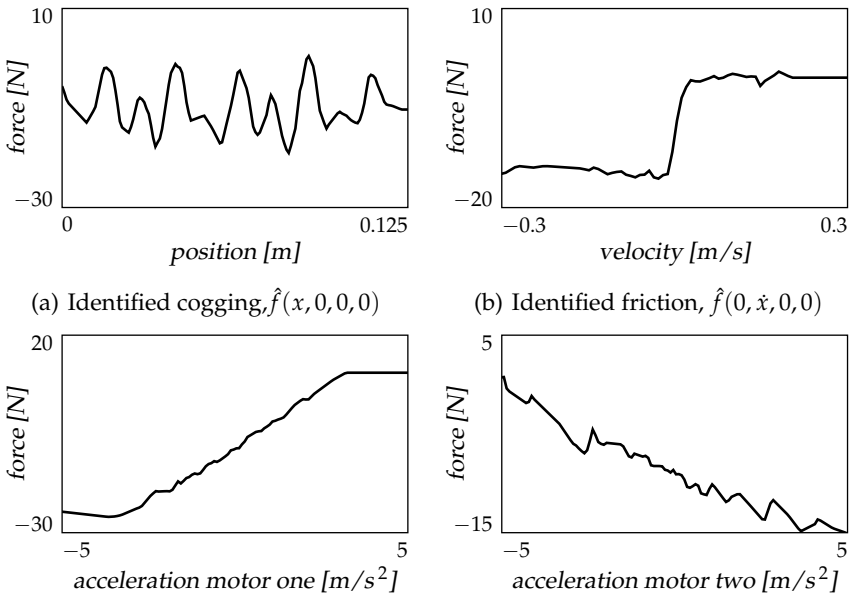
In figure 5.20 the output of the function approximator is plotted for the smallest MSE. As before, only one input alters in each plot, making



(a) # key samples

(b) RMS of tracking error, no learning: RMS = 182 [μm]

Figure 5.19: Two motors moving; influence of noise estimate for different inputs



(a) Identified cogging, $\hat{f}(x, 0, 0, 0)$

(b) Identified friction, $\hat{f}(0, \dot{x}, 0, 0)$

(c) Identified mass, $\hat{f}(0, 0, \ddot{x}, 0)$

(d) Identified dynamic coupling, $\hat{f}(0, 0, 0, \ddot{x})$

Figure 5.20: Identified effects

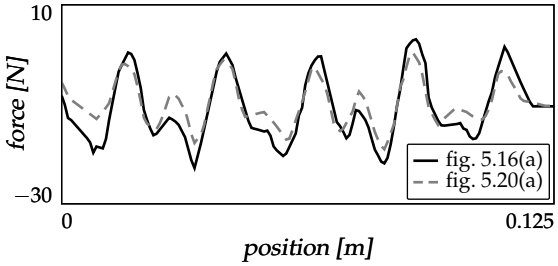


Figure 5.21: Identified cogging for one and two moving motors

it impossible to show state dependent effects that depend on several inputs. Approximating the dynamic coupling by a straight line results in a coupling of -1.5 [N/m s^{-2}]. The mass is approximated by 5.6 [kg].

Figure 5.21 shows the cogging force identified for one and two motors moving. Although the form is equivalent, there are differences between the effect identified. The reason of these differences is unknown, but approximation errors might play a role.

5.4.3 All motors moving

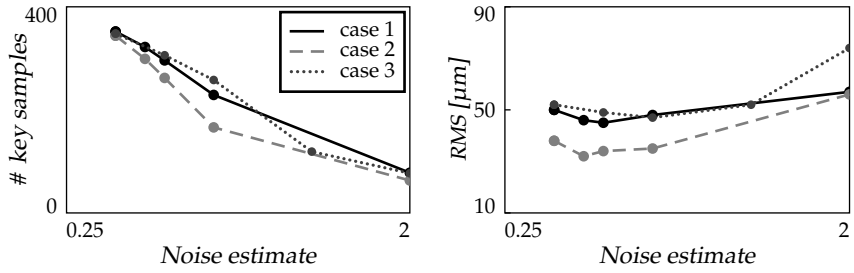
In the last set of experiments, all three motors moved independently. This set of experiments is similar to the previous set of experiments in which two motors were moving, because, apart from the motor characteristics, the dynamic coupling from the other motors had to be learnt too. As before, we consider only the tracking error of motor number one; the other motors are only used as disturbances and are therefore only controlled by a PD-controller.

As was seen for two moving motors, the effect of the dynamic coupling on the tracking error of motor number one, could be compensated for reasonably if the commanded accelerations of the other motors were added as inputs to the feedforward controller. Therefore, similar cases are considered in this set of experiments:

Case 1) Only the position, velocity and acceleration of motor number one are used as inputs. The dynamic coupling is neglected in this experiment. Therefore: $\mathbf{x} = (r_{x,1}, r_{v,1}, r_{a,1})$.

Case 2) Apart from the previous inputs, the commanded accelerations of both other motors are added as input: $\mathbf{x} = (r_{x,1}, r_{v,1}, r_{a,1}, r_{a,2}, r_{a,3})$.

Case 3) All the commanded states are used. This means that the position, velocity and acceleration of all motors are used as input; the



(a) Number of found key samples

(b) RMS of tracking error, no learning: RMS = 176 [μm]

Figure 5.22: Three motors moving; influence of noise estimate for different inputs

number of inputs becomes 9: $\mathbf{x} = (r_{x,1}, r_{v,1}, r_{a,1}, r_{x,2}, r_{v,2}, r_{a,2}, r_{x,3}, r_{v,3}, r_{a,3})$.

The results are summarised in figure 5.22. The results are similar to the results of the previous set of experiments: with only the three commanded states of motor number one, the dynamic coupling could not be compensated for; if the commanded accelerations of both other motors were included, the effect of the coupling on motor number one could be compensated for, which attenuated the tracking error; adding more inputs increased the tracking error again, because the extra inputs had little to no predictive power and complicated the approximation. The inputs needed for the compensation of the dynamic coupling and the motor characteristics are:

$$\mathbf{x} = (r_{x,1}, r_{v,1}, r_{a,1}, r_{a,2}, r_{a,3}); \tag{5.19}$$

the inputs with (too) little predictive power are:

$$\mathbf{x}_u = (r_{x_2}, r_{v_2}, r_{x_3}, r_{v_3}). \tag{5.20}$$

5.5 Review

In chapter 3 and 4 function approximators were introduced that showed good theoretical properties for learning control. In this chapter, these off- and on-line function approximators are included as part of the learning feedforward controller. This learning control scheme significantly reduced the tracking error for all the experiments on the Tripod. In the

experiments, the function approximator was tested with 3, 4, 5, 6 and 9 inputs.

Phase-corrected learning feedforward control was introduced in the beginning of this chapter as an improvement of traditional learning feedforward control. The phase-corrected variant corrects the phase lag, when the feedback signal is used to learn the state-dependent effects. This phase shift degenerates the learning accuracy and can even cause instability. By means of an experiment, it becomes clear that the phase corrected version of learning feedforward results in a smaller tracking error.

The phase-corrected LFFC was used as a control setting for an experiment in which KSM was used as an off-line function approximator. The KSM was capable of inferring the state-dependent effects from the data, by which the tracking error decreased significantly. Before learning, the tracking error of the PD-controlled system was 184 [µm], while after learning the error decreased to 22 [µm]. It can safely be concluded that the KSM is capable of learning state-dependent effects that depend on three inputs.

The recursive version of KSM, RKSM, was used in the *on-line* phase-corrected learning feedforward scheme as part of the feedforward controller. Each millisecond a new sample was supplied to the learning feedforward controller, which had to identify the state-dependent effects. The RKSM in the learning feedforward controller was tested for 3, 4, 5, 6 and even 9 inputs: significant error reduction was realised for all these tests.

Although a significant error reduction was realised in all the experiments, several inputs had little predictive power. Including them unnecessarily complicated the approximation, resulting in a larger tracking error.

The number of key samples that was needed for the error reduction kept manageable. This kept the calculation time limited. The calculations of the update were, however, performed outside the real-time loop, so that timing of the feedback controller was definitely correct.

Experiments for which the feedforward controller requires more than two inputs, are of no use with the BSN function approximator used in previous work. The number of B-splines becomes too large, and more important, the generalisation becomes too bad due to the limited support of the B-splines. Therefore, not substantial error reduction would result.

Tuning the function approximator is easy. Only one parameter has to be tuned with a clear physical interpretation: the noise level. This

gave a good approximation of the state-dependent effects, without the need for many attempts to find the correct parameter settings.

Convergence of the learning controller was fast. In 25 seconds of learning, the tracking error did not decrease significantly anymore.

The limitations of the learning control scheme were found to be due to the flexibilities. These dynamic effects cannot be compensated for by the current setup. If the control scheme is to be used in setups with considerable flexibilities, it is recommended that it is investigated what the influence of these vibrations are on the learning performance, and whether it is recommended to compensate for them by a modified version of LFFC.

Based on the experiments, it can be concluded that these function approximators can be used in a learning control setting where multi-dimensional approximations are required, and thus they solve the problem stated in chapter 1.

Six

Discussion

THE PURPOSE OF THIS RESEARCH was to find a function approximator that could be used as part of the Learning FeedForward Controller, and that was not as prone to the curse of dimensionality as the B-spline network, used in previous work on LFFC. This purpose led to the following problem definition:

Problem definition:

> *Find a function approximator that can be used in a learning control setting.*

In order to attain a function approximator that could be used in a learning control setting, a set of conditions was specified. As the function approximator was needed in an on-line as well as in an off-line learning control setting, not one, but two function approximators were searched for. The first had to find a relation within the data when this data was presented as a batch, while the second had to find a relation in the data when the data was presented as an endless stream.

First we will review the individual chapters, based on which the conclusions are drawn. Recommendations for further research conclude this thesis.

6.1 Review

Chapter 3, off-line function approximation

An off-line function approximator, called Key Sample Machine (KSM), has been developed that represents the data by a subset of the data. This approach is similar to the approach of the Support Vector Machine

(SVM). Because the prediction is made based on similarities with training samples, the input space is not necessarily divided into regions, as done by e.g. B-splines and the Radial Base Functions Network (RBFN), and the KSM is therefore less prone to the curse of dimensionality. The samples that are used to summarise the data are called *key samples*. The interpolation between the key samples is brought about by a dual indicator function, which is equal to the kernel function of the SVM. Because of the freedom to choose a kernel function, the class of functions that can be used for an approximation is not limited to one class, as is by RBFN or the Multilayer Perceptron (MLP).

In contrast with SVM, which uses an ϵ-insensitive cost function, a *quadratic cost function* has been used for KSM. This does not inherently result in a sparse solution. In Least Squares Support Vector Machines (LSSVM), which also uses a quadratic cost function, a sparse subset of training samples for prediction is found, by successively removing those samples from the training set that have little influence on the prediction. This pruning scheme omits valuable information by removing training samples. KSM uses a *subset selection scheme* to find the key samples that *summarise* the data set. In this procedure, all the training data is used to train the parameters accompanying each key sample. Because the subset selection is now an *explicit* step in the approximation scheme, a selection scheme that is appropriate for the problem on hand can be used. In this thesis the forward selection scheme was used. This scheme includes one key sample a time, until a good enough approximation is found. The calculation time required remains small when the number of key samples remains small.

A stopping criterion for the selection scheme has been introduced that tests whether the inclusion of an extra key sample is *statistically relevant.* Only if it is, the sample is included in the set of key samples to predict for new inputs. Because of the statistical test, the accuracy of the prediction is based on the *quality and quantity* of the data. As a result, the fitting of noise is unlikely.

The noise variance of the Gaussian noise that corrupts the targets is used to test whether the inclusion of an extra key sample is statistically relevant. This noise variance is often unknown an should be estimated. The noise estimate can also be interpret as a design parameter to trade off few key samples and an accurate prediction.

Evaluation of KSM and comparison of KSM with other off-line methods, showed that KSM could handle high-dimensional input spaces and that only a limited number of key samples are needed for a prediction. In comparison with other support vector based methods, it is rather fast and gives a smaller prediction error for noisy data.

Chapter 4, on-line function approximation

A recursive version of KSM has been developed for on-line learning control. This recursive function approximator is called Recursive Key Sample Machine (RKSM) and is capable of updating its *parameters* on-line as well as adapting its *structure.* The calculations necessary for each update, are recursive. The update for the weights of the key samples due to a new sample is exact. This means that there is no difference in the weights calculated based on all the samples, or calculated by updating for each new sample. Removal of a key sample from the set of key samples is also exact. However, the addition of a sample to the set of key samples is *not exact.* This update cannot be made exact, because the sample previous supplied would be required, and they are omitted. Therefore, it is assumed that the new key sample is uncorrelated with the current set of key samples. This is however, not true.

Evaluation of RKSM showed that it is capable of selecting a good set of key samples from the continuous data stream. The mean squared error (MSE) of the on-line version was nearly equal to the MSE of the off-line version. This has been realised by including an *omission scheme* that omits samples that have become superfluous. As the behaviour of the on- and off-line approximators is nearly the same, the handling of high-dimensional input spaces is also nearly equal. The on-line approximator can handle high-dimensional input spaces just as well as the off-line approximator.

Similar to KSM, the approximation is dependent on the *quality* of the data. If the noise variance is low, the approximation is precise, but for a large noise variance, the approximation is rough. If the noise level is estimated *too* low, the approximator starts fitting noise. The ridge regression scheme is not capable of reducing this noisy approximation, but by means of *structural regularisation,* the number of key samples can be limited, which results in a smoother approximation. A disadvantage of this regularisation scheme is that fast fluctuations can no longer be approximated.

Time-variant functions do not pose a problem for RKSM, when a *forgetting* mechanism is incorporated.

Chapter 5, experiments

The KSM as well as the RKSM have been used as a function approximator in the learning feedforward controller. The use of KSM was tested in an off-line setting because it required all the data in a batch, while RKSM was tested in an on-line learning controller setting. The application on

which the function approximators were tested was a Tripod. This is a mechanical setup consisting of three coupled linear motors.

The phase-corrected learning feedforward control scheme was used. The phase-corrected variant of Learning FeedForward Control (LFFC) corrects the phase lag present in traditional LFFC.

The KSM was capable of learning the state dependent effects out of the data, by which the tracking error decreased significantly. The effects the KSM compensated for, depended on three variables.

The recursive version of KSM, updated its approximation at each training sample that became available, i.e. each millisecond an update was calculated. The RKSM in the learning feedforward controller was tested for 3, 4, 5, 6 and even 9 inputs, by which it realised a significant error reduction. Although a significant error reduction was realised for all these inputs, several inputs had little predictive power. Including them unnecessarily complicated the approximation, resulting in a larger tracking error.

The use of the noise estimate to tune the accuracy of the function approximator was found to be intuitive. Only one parameter has to be tuned and this parameter has a clear physical interpretation. Good approximation of the state-dependent effects were found, without a large number of attempts to find the correct parameter settings.

Convergence of the learning controller was fast. After 25 seconds of learning, the tracking error did not decrease significantly anymore.

In the experiment, the limitations of the learning control scheme were found to be due to the limited mechanical stiffness. These dynamic effects could not be compensated for by the current setup.

6.2 Conclusions

The conclusions that can be drawn from this research concerning function approximators are:

- An on- and off-line function approximator have been constructed that can be used in a learning control setting. This has been experimentally validated, and these function approximators therefore comply with the problem definition.
- The sample-based approach is a useful approach to circumvent the curse of dimensionality. Because the prediction is made based on similarities with training samples, the input space is not necessarily divided into regions. The number of key samples that is needed to represent the data depends on the relation underlying the data. In

order to get a good approximation, *all* the training data should be used to *train* the parameters corresponding to the key samples.

- The statistical test to decide whether a sample will become a key sample, results in an accuracy of the approximation depending on the quality of the data, and in the case of an off-line approximator, it also depends on the quantity of the data. This makes the fitting of the current noise realisation unlikely. The statistical test also results in a design parameter intuitive in use: the noise level. With this design parameter, a trade off can be made between the accuracy and the number of key samples.

- Assigning a weight to each key sample makes it possible to indicate that a key sample represents a multitude of data. Using these weights, new samples can be incorporated in an existing approximation without causing problems. Furthermore, these weights can help to select which key sample can be removed.

- For RKSM, the update of the approximation is not exact when the structure of the approximation is extended. We have assumed, that the new key sample is uncorrelated with the present key samples. This assumption is generally not true, but is required because the true correlation cannot be calculated due to the omission of previous samples.

- Structural regularisation can be implemented by including a key sample only if it differs enough from the current set of key samples. This form of regularisation limits the number of key samples and so smoothens the approximation. A disadvantage is that fast fluctuations cannot be approximated anymore.

- A forgetting mechanism can be incorporated so that time-varying functions can be approximated. The forgetting mechanism is also useful to indicate that the structure, when only few samples are processed, is a preliminary one, and is therefore not certain.

- The computation time of both KSM and RKSM is limited. An approximation was found within seconds if 2 000 samples are supplied in a batch to the key sample machine. The calculation time for the on-line approximator was much smaller, because in the on-line approximator, the larger part of the calculations is spent on selecting the best candidate key sample, and this is not done for the off-line approximator.

As to the learning feedforward controller, it can be concluded that:

- The phase corrected learning feedforward controller with the recursive key sample machine as function approximator works good. For all the experiments a significant tracking error reduction was achieved.
- Phase correction for learning the state-dependent effects in a learning feedforward setting, increases the tracking performance.
- Learning feedforward control, as well as its phase-corrected variant, are incapable of compensating for dynamic effects, e.g. vibrations, of the plant. This limits the performance of the learning control scheme. The dynamics also limit the accuracy of the approximation of the state-dependent effects.

6.3 Recommendations for future work

The results found in this research may well be used in learning control. However, there remain several issues that require (more) research:

- In the on-line function approximator an omission scheme was introduced that removed the superfluous key samples. For the on-line approximator this is of more importance because fewer key samples imply less calculation time, which is restricted.

  However, in the off-line case an omission scheme can also be incorporated. This is expected to lead to fewer key samples for the same accuracy. The step-wise indicator selection scheme of Miller (1990) can be used to implement this.

- In this thesis the approximation was either made based on a batch of data or on one sample a time. However, the approximation scheme can be slightly altered so that *any* number of samples can be incorporated into an existing approximation. This can be useful, if the data becomes available as a sequence of small batches. It could also be used to supply the on-line function approximator with a priori knowledge. This intermediate form can form the connection between the on- and off-line approximators.

  The weights on the key samples are used to indicate that the key samples represent a multitude of data. As a result the inclusion of a sample in the approximation can be done appropriately. In section 4.2.2 the approximation update is done for one sample, but this can be done repeatedly to include more samples.

The selection of a candidate sample that can become a key sample can be done in the same way as the selection in the off-line case by finding the sample which corresponds most with the residual *of the new batch of data.* If the new batch only contains one sample, and is therefore equal to the on-line case, this selection of the candidate is trivial.

Finally a test has to be done to see if this candidate sample should become a key sample. Again, one can test if the corresponding parameter is likely to be zero.

- In this thesis the selection of inputs to the approximator was not explicitly considered. However, if the number of inputs increases, it might be useful to incorporate the selection of the inputs in the approximation problem. One might think, for instance, of principle component analysis or projection pursuit (Cao, Chua, Chong, Lee, and Gu, 2003; Hall, 1989; Haykin, 1994). In the experiments we found that inclusion of extra inputs with little predictive power worsens the approximation and therefore an input selection scheme can give an improvement.

- The approximation scheme assumed that the *output* was contaminated with noise, but the *input* was not. However, because the states commanded were used as input of the function approximator, instead of the actual states, the input also contained errors. The total least squares method takes into account that the inputs are also corrupted, e.g. (Björck, 1996). In Huffel and Vandewalle (1991) it is shown that the prediction accuracy increases if total least squares is used, which makes it worth investigating if this scheme can give a tracking error reduction in learning feedforward control.

- The results of the approximation in the experiments showed a non-smooth approximation (figure 5.20). In section 4.2.2 the correlation between the key samples was assumed to be non-existing. However, this correlation might be used to implement a form of *spatial filtering*. With this filtering, the approximation might be smoothed. Appendix C treats the existence of a real-valued indicator vector accompanying the choice of the correlation and might be a starting point for implementing spatial filtering.

- For the construction of the learning filter L (5.3), the Tripod was modelled as a moving mass, thus neglecting a part of the dynamics. As a result, the vibrations in the error signal were supplied to the learning mechanism as a state-dependent effect to be learnt. It

is recommended to investigate how the vibrations, and other unmodelled dynamics, influence the learning behaviour. Design of a learning controller by means of an H_∞ criterion can be considered (De Roover and Bosgra, 2000).

- We assumed that the state-dependent effects acted on the input of the plant and we could therefore compensate for them by applying an opposite force to the input of the plant (figure 1.2(a)). If the location where we act is different from the location that is disturbed, e.g. the printer head in the introduction, then this assumption does no longer hold. However, as long as the connection between the locations is stiff, with regard to the frequency by which the disturbance needs to be compensated for, the current LFFC setup can be used. If the LFFC can be extended so that the connection does not need to be stiff, then the range of applications extends.
- In figure (5.15) we saw that the tracking error did not decrease for frequencies larger than 25 [Hz]. In a set of experiments it was found that the bandwidth of the controller and of the low-pass filters had influence on the error-reduction frequency range. It is interesting to investigate *why* the tracking error did not decrease after 25 [Hz] and with this insight try to increase the error-reduction frequency range.

Appendix A

Optimisation

OPTIMISATION THEORY treated in this appendix is limited to the theory used in this thesis. For more information concerning optimisation, refer, among others, to (Aoki, 1971; Boyd and Vandenberghe, 2004).

A.1 With equality constraints

A convex minimisation problem with equality constraints on a convex set can be solved by use of Lagrange's theory. First a Lagrangian function has to be defined (Cristianini and Shawe-Taylor, 2000, ch. 5):

Definition 3 (Lagrangian function) *Given an optimisation problem with objective function $f(\mathbf{b})$, and equality constraints $h_i(\mathbf{b}) = 0, i = 1, \ldots, m$, we define the Lagrangian function as:*

$$\mathcal{L}(\mathbf{b}, \boldsymbol{\alpha}) = f(\mathbf{b}) + \sum_{i=1}^{m} \alpha_i h_i(\mathbf{b}), \tag{A.1}$$

where the coefficients α_i are called the Lagrange multipliers.

With this Lagrangian function, we can find necessary and sufficient conditions for the solution $\mathbf{b}^*$ to be optimal.

Theorem 1 (Lagrange) *A necessary condition for a normal point $\mathbf{b}^*$ to be a minimum of $f(\mathbf{b})$ subject to $h_i(\mathbf{b}) = 0, i = 1, \ldots, m$, with $f, h_i \in C^1$, is:*

$$\frac{\partial \mathcal{L}(\mathbf{b}^*, \boldsymbol{\alpha}^*)}{\partial \mathbf{b}} = 0, \tag{A.2a}$$

$$\frac{\partial \mathcal{L}(\mathbf{b}^*, \boldsymbol{\alpha}^*)}{\partial \boldsymbol{\alpha}} = 0, \tag{A.2b}$$

for some values of $\boldsymbol{\alpha}^*$*. The above conditions are also sufficient provided that* $\mathcal{L}(\mathbf{b}^*, \boldsymbol{\alpha}^*)$ *is a convex function of* $\mathbf{b}$*.*

Least Squares Support Vector Machine

The solution of the optimisation problem as used by LSSVM can be found with the above-given theory. The solution of LSSVM is closely related to the dual least squares treated in section 3.1.2. The optimisation problem for LSSVM is:

$$\min_{\mathbf{b},b_0,\mathbf{e}} \tfrac{1}{2}\mathbf{e}^{\mathrm{T}}\mathbf{e} + \tfrac{\lambda}{2}\mathbf{b}^{\mathrm{T}}\mathbf{b} \tag{A.3}$$

such that

$$\mathbf{y} - \mathbf{X}\mathbf{b} - b_0\mathbf{1} - \mathbf{e} = 0, \tag{A.4}$$

in which $\mathbf{1}$ is a column-vector containing 1's. This optimisation problem is only different from the dual least squares by not including the offset term in the regularisation term, see (3.13). Furthermore, the offset term is explicitly denoted in (A.4), while it is incorporated in $\mathbf{X}$ in (3.13).

A Lagrangian can be constructed for this optimisation problem:

$$\mathcal{L} = \tfrac{1}{2}\mathbf{e}^{\mathrm{T}}\mathbf{e} + \tfrac{\lambda}{2}\mathbf{b}^{\mathrm{T}}\mathbf{b} + \boldsymbol{\alpha}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\mathbf{b} - b_0\mathbf{1} - \mathbf{e}). \tag{A.5}$$

Differentiation of the Lagrangian yields the conditions necessary for optimality:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{b}} &= \lambda\mathbf{b} - \mathbf{X}^{\mathrm{T}}\boldsymbol{\alpha} = 0, && \text{(A.6a)}\\
\frac{\partial \mathcal{L}}{\partial b_o} &= \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{1} = 0, && \text{(A.6b)}\\
\frac{\partial \mathcal{L}}{\partial \mathbf{e}} &= \mathbf{e} - \boldsymbol{\alpha} = 0, && \text{(A.6c)}\\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\alpha}} &= \mathbf{y} - \mathbf{X}\mathbf{b} - b_0\mathbf{1} - \mathbf{e} = 0. && \text{(A.6d)}
\end{aligned}$$

Elimination of $\mathbf{b}$ and $\mathbf{e}$ through substitution and rewriting this set of equations in matrix form, results in:

$$\left[\begin{array}{c|c} 0 & \mathbf{1}^{\mathrm{T}} \\ \hline \mathbf{1} & \frac{1}{\lambda}\mathbf{X}\mathbf{X}^{\mathrm{T}} + \mathbf{I} \end{array}\right] \left[\begin{array}{c} b_0 \\ \hline \boldsymbol{\alpha} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \mathbf{y} \end{array}\right]. \tag{A.7}$$

In order to be compatible with the standard notation of LSSVM, we use $\tilde{\boldsymbol{\alpha}} = \boldsymbol{\alpha}/\lambda$. This alters (A.7) to:

$$\left[\begin{array}{c|c} 0 & \mathbf{1}^{\mathrm{T}} \\ \hline \mathbf{1} & \mathbf{X}\mathbf{X}^{\mathrm{T}} + \lambda \mathbf{I} \end{array}\right] \left[\begin{array}{c} b_0 \\ \hline \tilde{\boldsymbol{\alpha}} \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \mathbf{y} \end{array}\right]. \tag{A.8}$$

The solution of LSSVM is equal to the solution of dual least squares (3.19a). However, for LSSVM it simultaneously has to hold, that the sum of the Lagrangian multipliers is zero.

A.2 With inequality constraints

The Kuhn-Tucker theorem gives conditions for an optimal solution to a general optimisation problem (Cristianini and Shawe-Taylor, 2000, ch. 5).

Theorem 2 (Kuhn-Tucker) *Given an optimisation problem with convex domain $\Omega \subseteq \mathbb{R}^n$,*

$$\begin{array}{lll} \textit{minimise} & f(\mathbf{b}), & \mathbf{b} \in \Omega, \\ \textit{subject to} & g_i(\mathbf{b}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{b}) = 0, & i = 1, \ldots, m, \end{array} \tag{A.9}$$

with $f \in C^1$ convex and g_i and h_i affine, necessary and sufficient conditions for a normal point $\mathbf{b}^*$ to be an optimum are the existence of $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ such that

$$\frac{\partial \mathcal{L}(\mathbf{b}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{b}} = 0, \tag{A.10a}$$

$$\frac{\partial \mathcal{L}(\mathbf{b}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0, \tag{A.10b}$$

$$\alpha_i^* g_i(\mathbf{b}^*) = 0, \quad i = 1, \ldots, k, \tag{A.10c}$$

$$g_i(\mathbf{b}^*) \leq 0, \quad i = 1, \ldots, k, \tag{A.10d}$$

$$\alpha_i^* \geq 0, \quad i = 1, \ldots, k. \tag{A.10e}$$

In this theorem, $\mathcal{L}$ is the generalised Lagrangian function:

$$\mathcal{L}(\mathbf{b}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{b}) + \boldsymbol{\alpha}^{\mathrm{T}} \mathbf{g}(\mathbf{b}) + \boldsymbol{\beta}^{\mathrm{T}} \mathbf{h}(\mathbf{b}), \tag{A.11}$$

while the conditions $\alpha_i^* g_i(\mathbf{b}^*) = 0, i = 1, \ldots, k$ are known as the Karush-Kuhn-Tucker (KKT) conditions. These conditions state that if a constraint is active, then the multiplier is equal or larger than zero, while for inactive constraints the multiplier is equal to zero.

Support Vector Machine

The support vector machine optimisation problem can be reformulated with the above-given theory to a dual problem. Repeating the optimisation problem of section 3.1:

$$\min_{\mathbf{b},b_0} C \sum_{i=1}^{N} (\xi_i + \xi_i^*) + \tfrac{1}{2}\mathbf{b}^{\mathrm{T}}\mathbf{b} \tag{A.12}$$

with the constraints:

$$\begin{aligned}
\mathbf{f}(\mathbf{x}_i)\mathbf{b} + b_0 - y_i &\leq \epsilon + \xi_i, && \text{(A.13a)}\\
y_i - \mathbf{f}(\mathbf{x}_i)\mathbf{b} - b_0 &\leq \epsilon + \xi_i^*, && \text{(A.13b)}\\
-\xi_i^* &\leq 0, && \text{(A.13c)}\\
-\xi_i &\leq 0. && \text{(A.13d)}
\end{aligned}$$

This formulation is equal to the standard formulation and a generalised Lagrangian can be formulated:

$$\begin{aligned}
\mathcal{L} = {} & C \sum_{i=1}^{N} (\xi_i + \xi_i^*) + \tfrac{1}{2}\mathbf{b}^{\mathrm{T}}\mathbf{b} \\
& + \sum_{i=1}^{N} \alpha_i^* (y_i - \mathbf{f}(\mathbf{x})\mathbf{b} - b_0 - \epsilon - \xi_i^*) \\
& + \sum_{i=1}^{N} \alpha_i (\mathbf{f}(\mathbf{x})\mathbf{b} + b_0 - y_i - \epsilon - \xi_i) \\
& - \sum_{i=1}^{N} \gamma_i \xi_i + \gamma_i^* \xi_i^*.
\end{aligned} \tag{A.14}$$

In this equation, $\alpha_i^{(*)}$ and $\gamma_i^{(*)}$ are the Lagrangian multipliers that correspond to the inequality constraints. This Lagrangian has to be minimised for the primal variables, while it has to be maximised for the

Lagrangian multipliers. Differentiation of the Lagrangian results in:

$$\frac{\partial \mathcal{L}}{\partial b_0} = \sum_i \alpha_i - \sum_i \alpha_i^* = 0 \quad \Rightarrow \sum_i \alpha_i = \sum_i \alpha_i^* \tag{A.15a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \mathbf{b} + \sum_i \alpha_i \mathbf{f}^{\mathrm{T}}(\mathbf{x}_i) - \sum_i \alpha_i^* \mathbf{f}^{\mathrm{T}}(\mathbf{x}_i) = 0 \Rightarrow \mathbf{b} = \sum_i (\alpha_i^* - \alpha_i)\mathbf{f}^{\mathrm{T}}(\mathbf{x}_i)$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \alpha_i - \gamma_i = 0 \quad \Rightarrow \gamma_i = C - \alpha_i \tag{A.15c}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i^*} = C - \alpha_i^* - \gamma_i^* = 0 \quad \Rightarrow \gamma_i^* = C - \alpha_i^* \tag{A.15d}$$

The second of this set of equations relates the parameter vector with the Lagrangian multipliers. Substitution of these relations into the Lagrangian yield the dual optimisation problem:

$$\max_{\boldsymbol{\alpha},\boldsymbol{\alpha}^*} \sum_{i=1}^{N} (\alpha_i^* - \alpha_i)\, y_i - \epsilon \sum_{i=1}^{N} (\alpha_i^* + \alpha_i) - \frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i^* - \alpha_i)\,(\alpha_j^* - \alpha_j)\mathbf{f}(\mathbf{x}_i)\mathbf{f}^{\mathrm{T}}(\mathbf{x}_k). \tag{A.16}$$

For a feasible solution, the following constraints need to hold:

$$0 \leq \alpha_i \leq C, \tag{A.17a}$$

$$0 \leq \alpha_i^* \leq C, \tag{A.17b}$$

$$\sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0. \tag{A.17c}$$

At the optimal solution, the KKT conditions hold:

$$\alpha_i(\mathbf{f}(\mathbf{x}_i)\mathbf{b} + b_0 - y_i - \epsilon - \xi_i) = 0, \tag{A.18a}$$

$$\alpha_i^*(y_i - \mathbf{f}(\mathbf{x}_i)\mathbf{b} - b_0 - \epsilon - \xi_i^*) = 0, \tag{A.18b}$$

$$\xi_i^* \xi_i = 0, \tag{A.18c}$$

$$\alpha_i^* \alpha_i = 0, \tag{A.18d}$$

$$(\alpha_i - C)\xi_i = 0, \tag{A.18e}$$

$$(\alpha_i^* - C)\xi_i^* = 0. \tag{A.18f}$$

The optimal values of $\boldsymbol{\alpha}^{(*)}$ can be found by the algorithms described in (Boyd and Vandenberghe, 2004; Keerthi et al., 2001) or the software of (Chang and Lin, 2003) can be directly used.

Appendix B

Implementation

THE CALCULATIONS OF THE algorithms presented in this thesis can be done recursively. This appendix treats an implementation for the KSM and the RKSM.

B.1 Key Sample Machine

The complete algorithm for the KSM is given in algorithm 3.1 on page 59. Implementation of this algorithm has not been treated, and in this section an implementation of the algorithm by means of a QR decomposition is treated. Knowledge on decompositions and matrix properties is assumed. For more information on these subjects, we refer to (Björck, 1996; Golub and Van Loan, 1996; Stewart, 1998, 2001).

First, it is treated how the OLS problem can be solved by means of the QR decomposition, while second, the inclusion of a key sample is treated. Other steps of the algorithm are readily implemented. The QR decomposition is used for the calculations due to its numerical stability (Björck, 1996; Stewart, 1998).

B.1.1 Background

The OLS problem as stated in section 3.1.1:

$$\min_{\mathbf{b}} \|\mathbf{X}\mathbf{b} - \mathbf{y}\|_2^2, \tag{B.1}$$

can be solved by means of theory from linear algebra with theorem 2.1 of Stewart (1998):

Theorem 3 *Let* $\mathbf{X}$ *be of full column rank and have a* QR *decomposition of the form:*

$$\mathbf{X} = \begin{bmatrix}\mathbf{Q}_\mathbf{X} & \mathbf{Q}_\perp\end{bmatrix} \begin{bmatrix}\mathbf{R} \\ 0\end{bmatrix}. \tag{B.2}$$

Then the solution of the least squares problem of minimising $\|\mathbf{y} - \mathbf{X}\mathbf{b}\|_2^2$ *is uniquely determined by the* QR *equation*

$$\mathbf{R}\mathbf{b} = \mathbf{Q}_\mathbf{X}^\mathrm{T}\mathbf{y} \equiv \mathbf{z}_\mathbf{X}. \tag{B.3}$$

The least squares approximation of $\mathbf{y}$ *is given by*

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{b} = \mathbf{P}_\mathbf{X}\mathbf{y} = \mathbf{Q}_\mathbf{X}\mathbf{Q}_\mathbf{X}^\mathrm{T}\mathbf{y} = \mathbf{Q}_\mathbf{X}\mathbf{z}_\mathbf{X}. \tag{B.4}$$

The residual vector is given by

$$\mathbf{r} = \mathbf{y} - \mathbf{X}\mathbf{b} = \mathbf{P}_\perp\mathbf{y} = \mathbf{Q}_\perp\mathbf{Q}_\perp^\mathrm{T}\mathbf{y} \equiv \mathbf{Q}_\perp\mathbf{z}_\perp, \tag{B.5}$$

and the residual sum of squares is

$$\|\mathbf{r}\|_2^2 = \|\mathbf{z}_\perp\|_2^2. \tag{B.6}$$

Moreover, the residual at the minimum is orthogonal to the column space of $\mathbf{X}$.

In this theorem, $\mathbf{P}_\mathbf{X}$ is the orthogonal projection on the space spanned by the columns of $\mathbf{X}$. It is interesting to note that $\mathbf{P}_\mathbf{X} = \mathbf{X}(\mathbf{X}^\mathrm{T}\mathbf{X})^{-1}\mathbf{X}^\mathrm{T}$, which relates the projection matrix to the prediction found by the normal equations.

In order to calculate $\mathbf{z}_\mathbf{X}$, the following theorem can be used (Stewart, 1998, Theorem 2.2.):

Theorem 4 *Let the* QR *factorisation of the matrix* $\begin{bmatrix}\mathbf{X} & \mathbf{x}\end{bmatrix}$ *be partitioned in the form*

$$\begin{bmatrix}\mathbf{X} & \mathbf{x}\end{bmatrix} = \begin{bmatrix}\mathbf{Q} & \mathbf{q}\end{bmatrix} \begin{bmatrix}\mathbf{R} & \mathbf{r} \\ 0 & \rho\end{bmatrix}. \tag{B.7}$$

Then

$$\rho\mathbf{q} = \mathbf{P}_\mathbf{X}^\perp\mathbf{x} \quad \textit{and} \quad \mathbf{r} = \mathbf{Q}^\mathrm{T}\mathbf{x}. \tag{B.8}$$

In terms of our least squares problem, this theorem states that for an *augmented indicator matrix:* $\mathbf{X}_\mathrm{a} = \begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}$, the following holds:

$$\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} = \begin{bmatrix}\mathbf{Q} & \mathbf{r}/\|\mathbf{r}\|_2\end{bmatrix} \begin{bmatrix}\mathbf{R} & \mathbf{z}_\mathbf{X} \\ 0 & \|\mathbf{r}\|_2\end{bmatrix}. \tag{B.9}$$

The parameter vector $\mathbf{b}$ is calculated by solving $\mathbf{Rb} = \mathbf{z_X}$, while the $\mathbf{z_X}$ and $\mathbf{R}$ result from the QR decomposition of the augmented indicator matrix.

The $\mathbf{R}$ matrix of the QR decomposition can be found by means of a Cholesky decomposition, without the necessity to calculate the corresponding $\mathbf{Q}$. When $\mathbf{X}$ has a full column rank, then the Cholesky decomposition of $\mathbf{X}^T\mathbf{X}$ exists:

$$\mathbf{X}^T\mathbf{X} = \mathbf{R}_{chol}^T\mathbf{R}_{chol}, \tag{B.10}$$

in which $\mathbf{R}_{chol}$ is upper triangular. Substitution of the QR decomposition for $\mathbf{X}$ in $\mathbf{X}^T\mathbf{X}$ yields:

$$\mathbf{X}^T\mathbf{X} = \mathbf{R}^T\mathbf{Q}^T\mathbf{QR} = \mathbf{R}^T\mathbf{R}. \tag{B.11}$$

From which it follows that the Cholesky decomposition of $\mathbf{X}^T\mathbf{X}$ equals the $\mathbf{R}$ of the QR decomposition of $\mathbf{X}$, because both are upper triangular. The matrix $\mathbf{Q}$ is not needed in any of these calculations and can be omitted to save memory space.

So, in order to solve the OLS problem, we first create an augmented indicator matrix (B.9). By means of a Cholesky decomposition (B.10), the vector $\mathbf{z_X}$ and the matrix $\mathbf{R}$ are found. With these, the parameter vector $\mathbf{b}$ can be calculated (B.3).

B.1.2 Inclusion of a key sample

When a key sample is added to the set of present key samples, the augmented decomposition has to be refreshed. A similar approach as in section 4.2.2 is used.

Before the inclusion of a new key sample, the augmented indicator matrix and its product are given as:

$$\mathbf{X}_a = \begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} \quad \Rightarrow \quad \mathbf{X}_a^T\mathbf{X}_a = \begin{bmatrix}\mathbf{X}^T\mathbf{X} & \mathbf{X}^T\mathbf{y} \\ \mathbf{y}^T\mathbf{X} & \mathbf{y}^T\mathbf{y}\end{bmatrix}. \tag{B.12}$$

The corresponding augmented Cholesky decomposition is given as:

$$\mathbf{R}_a = \begin{bmatrix}\mathbf{R} & \mathbf{r} \\ 0 & \rho\end{bmatrix} \quad \Rightarrow \quad \mathbf{R}_a^T\mathbf{R}_a = \begin{bmatrix}\mathbf{R}^T\mathbf{R} & \mathbf{R}^T\mathbf{r} \\ \mathbf{r}^T\mathbf{R} & \mathbf{r}^T\mathbf{r} + \rho^2\end{bmatrix}. \tag{B.13}$$

Because $\mathbf{X}_a^T\mathbf{X}_a = \mathbf{R}_a^T\mathbf{R}_a$, the following holds before the inclusion:

$$\mathbf{R}^T\mathbf{R} = \mathbf{X}^T\mathbf{X}, \tag{B.14a}$$

$$\mathbf{R}^T\mathbf{r} = \mathbf{X}^T\mathbf{y}, \tag{B.14b}$$

$$\mathbf{y}^T\mathbf{y} = \mathbf{r}^T\mathbf{r} + \rho^2. \tag{B.14c}$$

The new key sample induces an indicator function, and the values of this function for all the training samples are contained in the vector $\mathbf{x}$. Because of the inclusion of the key sample, the updated matrices are partitioned as follows:

$$\bar{\mathbf{X}}_a = \begin{bmatrix} \mathbf{X} & \mathbf{x} & \mathbf{y} \end{bmatrix}, \quad \bar{\mathbf{R}}_a = \begin{bmatrix} \bar{\mathbf{R}} & \mathbf{g} & \bar{\mathbf{r}} \\ 0 & \gamma & \bar{\rho} \\ 0 & 0 & \tau \end{bmatrix}. \tag{B.15}$$

After the update, $\bar{\mathbf{X}}_a^T \bar{\mathbf{X}}_a = \bar{\mathbf{R}}_a^T \bar{\mathbf{R}}_a$ still has to hold:

$$\begin{bmatrix} \mathbf{X}^T\mathbf{X} & \mathbf{X}^T\mathbf{x} & \mathbf{X}^T\mathbf{y} \\ \mathbf{x}^T\mathbf{X} & \mathbf{x}^T\mathbf{x} & \mathbf{x}^T\mathbf{y} \\ \mathbf{y}^T\mathbf{X} & \mathbf{y}^T\mathbf{x} & \mathbf{y}^T\mathbf{y} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{R}}^T\bar{\mathbf{R}} & \bar{\mathbf{R}}^T\mathbf{g} & \bar{\mathbf{R}}^T\bar{\mathbf{r}} \\ \mathbf{g}^T\bar{\mathbf{R}} & \mathbf{g}^T\mathbf{g} + \gamma^2 & \mathbf{g}^T\bar{\mathbf{r}} + \gamma\bar{\rho} \\ \bar{\mathbf{r}}^T\bar{\mathbf{R}} & \bar{\mathbf{r}}^T\mathbf{g} + \gamma\bar{\rho} & \bar{\mathbf{r}}^T\bar{\mathbf{r}} + \bar{\rho}^2 + \tau^2 \end{bmatrix}. \tag{B.16}$$

Thus:

$$\bar{\mathbf{R}}^T\bar{\mathbf{R}} = \mathbf{X}^T\mathbf{X}, \tag{B.17a}$$
$$\bar{\mathbf{R}}^T\mathbf{g} = \mathbf{X}^T\mathbf{x}, \tag{B.17b}$$
$$\bar{\mathbf{R}}^T\bar{\mathbf{r}} = \mathbf{X}^T\mathbf{y}, \tag{B.17c}$$
$$\mathbf{g}^T\mathbf{g} + \gamma^2 = \mathbf{x}^T\mathbf{x}, \tag{B.17d}$$
$$\mathbf{g}^T\bar{\mathbf{r}} + \gamma\bar{\rho} = \mathbf{x}^T\mathbf{y}, \tag{B.17e}$$
$$\bar{\mathbf{r}}^T\bar{\mathbf{r}} + \bar{\rho}^2 + \tau^2 = \mathbf{y}^T\mathbf{y}. \tag{B.17f}$$

Which, in combination of the equalities in (B.14), give:

$$\bar{\mathbf{R}} = \mathbf{R}, \tag{B.18a}$$
$$\mathbf{g} = \mathbf{R}^{-T}\mathbf{X}^T\mathbf{x}, \tag{B.18b}$$
$$\bar{\mathbf{r}} = \mathbf{r}, \tag{B.18c}$$
$$\gamma^2 = \mathbf{x}^T\mathbf{x} - \mathbf{g}^T\mathbf{g}, \tag{B.18d}$$
$$\bar{\rho} = \tfrac{1}{\gamma}\left(\mathbf{x}^T\mathbf{y} - \mathbf{g}^T\mathbf{r}\right), \tag{B.18e}$$
$$\tau^2 = \mathbf{y}^T\mathbf{y} - \mathbf{r}^T\mathbf{r} - \bar{\rho}^2. \tag{B.18f}$$

By which the augmented decomposition of $\mathbf{R}_a$, (B.15), can be determined, and hence, the new value of $\mathbf{b}$. For the recursive calculations, $\mathbf{X}$ and $\mathbf{R}_a$ need to be stored.

Interesting to see, is that by using $\mathbf{R}^{-T}\mathbf{X}^T = \mathbf{Q}^T$, $\mathbf{g}$ of (B.18b) is the projection of the new column $\mathbf{x}$ on the space spanned by the columns of $\mathbf{Q}$. An other observation is that in these equations, γ is the norm of the residual of the new indicator vector after projection on the present indicators. If this value is too small, it is argued in section 4.2.4, that the indicator should not be included due to a lack of information.

The norm of the residual before the update is given as ρ, while after the update it becomes τ. Based on their difference, the statistical test (3.53) can be performed.

In these calculations the matrix $\mathbf{Q}$ is not stored to save memory space. The price to pay is an increase in the number of calculations to calculate

Table B.1: Relation between matrices used for the RKSM algorithm and implementation

matrix	decomposition
$\mathbf{R}$	$\mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{X}_{\mathrm{ks}} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{P}^{-\mathrm{T}}\mathbf{P}^{-1}\mathbf{X}_{\mathrm{ks}}$
$\mathbf{R}_{\mathrm{ks}}$	$\mathbf{R}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{R}_{\mathrm{ks}} = \mathbf{X}_{\mathrm{ks}}$
$\mathbf{z}_{\mathbf{X}}$	$\mathbf{z}_{\mathbf{X}} = \mathbf{Q}_{\mathbf{X}}^{\mathrm{T}}\mathbf{y}$
$\mathbf{P}$	$\mathbf{P}^{\mathrm{T}}\mathbf{P} = \mathbf{V}^{-1}$

the projection of a new vector (B.18b). It should be noted, that $\mathbf{R}$ is upper triangular, so the calculations for $\mathbf{g}$ are not excessive. One might argue that $\mathbf{Q}$ can be stored instead of $\mathbf{X}$ for these calculations. For the inclusion of the new key sample this is true, however, to calculate which indicator should be included next, line 2 and 5 of algorithm 3.1, the matrix $\mathbf{X}$ is required. Further investigation on the trade off between storing $\mathbf{Q}$ and the extra calculation time, might be worth investigating for future implementations to improve the calculation time.

B.2 Recursive Key Sample Machine

For RKSM three update algorithms need to be implemented. These algorithms are given in algorithm 4.1 on page 91. As with KSM, the QR decomposition is used. The calculations for the implementation are all performed on decompositions of the matrices used in the algorithm. Table B.1 relates the decompositions with the matrices of the algorithm. With $\mathbf{R}$ and $\mathbf{z}_{\mathbf{X}}$ the parameter vector $\mathbf{b}$ can be calculated as done for KSM. The targets of the key samples, $\mathbf{y}_{\mathrm{ks}}$, are not stored explicitly, because they are not required for prediction. If required, one can calculate them by use of the parameter vector.

The weight matrix $\mathbf{V}^{-1}$ can be updated whenever a new training sample becomes available, or it can be calculated from $\mathbf{X}_{\mathrm{ks}}$ and $\mathbf{X}$ when required. In approximation experiments, it was found that the time difference was not significant between these approaches. However, both methods are treated.

B.2.1 Updating the key samples

The first situation considered is the adaption of the key samples due to a newly supplied training sample. This update is treated in table 4.1(a).

As the number of key samples does not alter, the matrix $\mathbf{R}_{\text{ks}}$ does not change; $\mathbf{R}, \mathbf{z}_{\boldsymbol{\chi}}$ and $\mathbf{P}$, however, do. The matrix $\mathbf{R}$ and $\mathbf{z}_{\boldsymbol{\chi}}$ are updated simultaneously, and this is treated first.

The augmented indicator matrix is extended by a row due to the new training sample.

$$\mathbf{X}_{\text{a}} = \begin{bmatrix} \mathbf{X} & \mathbf{y} \end{bmatrix} \quad \Rightarrow \quad \mathbf{X}_{\text{a}} = \begin{bmatrix} \mathbf{X} & \mathbf{y} \\ \mathbf{f} & y \end{bmatrix}. \tag{B.19}$$

In this, $\mathbf{f}$ is the indicator vector and y is the new target. The matrix $\mathbf{X}$ and $\mathbf{y}$ are not kept in memory, but their decomposition and projection, $\mathbf{R}$ and $\mathbf{z}_{\boldsymbol{\chi}}$ respectively, are. The alterations for these decompositions, due to the new row in the augmented indicator matrix, are calculated simultaneously. For this, $\mathbf{R}, \mathbf{z}_{\boldsymbol{\chi}}, \mathbf{f}$ and y are combined into a matrix:

$$\begin{bmatrix} \mathbf{R} & \mathbf{z}_{\boldsymbol{\chi}} \\ \mathbf{f} & y \end{bmatrix}, \tag{B.20}$$

in which $\mathbf{R}$ is upper triangular. By means of e.g. (fast) Givens rotations, this matrix can be reduced to the following form:

$$\begin{bmatrix} \bar{\mathbf{R}} & \bar{\mathbf{z}}_{\boldsymbol{\chi}} \\ 0 & \varsigma \end{bmatrix}, \tag{B.21}$$

in which $\bar{\mathbf{R}}$ is again upper triangular. The algorithm to arrive at (B.21) from (B.20) is described in detail in Golub and Van Loan (1996, chapter 12) and Stewart (1998, chapter 4.3) and is therefore not treated here. In the updated matrix, a bar above a matrix denotes the updated version.

When (B.9) and (B.20) are compared, then it can be seen that $\mathbf{r}$ of (B.9) is not present in (B.20). The matrix of (B.20) has the following partitioning:

$$\left[\begin{array}{cccc|c} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ \hline \times & \times & \times & \times & \times \end{array}\right], \tag{B.22}$$

in which $\times$ stands for a non-zero element. Because the norm of the residual is not included in this partitioned matrix, the last Givens rotation to update the residual is not done. Therefore, ς is not the norm of the residual, but the part that should be incorporated to the residual norm, i.e.

ς^2 is the change in summed squared error due to the new sample. This change in error encompasses the prediction error of the new sample as well as the change in error of the old samples due to the change in the approximation.

Next to the decomposition of $\mathbf{X}$, $\mathbf{R}$ is also the decomposition of weighted key samples: $\mathbf{P}^{-1}\mathbf{X}_{\text{ks}}$. ς^2 can therefore be interpreted as the *weighted* change of the key samples *plus* the residual of the new sample. This is the preferred interpretation, because of the assumption that the new key sample is uncorrelated with the present key samples, assumption 4, one is incapable of calculating the increase of summed squared error of all the previous samples once a new key sample is added. The weighted change of the key samples plus the residual can be used for the test in (4.55).

The update of the weight matrix is calculated next. The update can be calculated by:

$$\mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{l} = \mathbf{f}^{\text{T}}, \tag{B.23a}$$

$$\bar{\mathbf{V}}^{-1} = \mathbf{V}^{-1} + \mathbf{l}^{\text{T}}\mathbf{l}, \tag{B.23b}$$

see algorithm 4.1(a). The calculation of $\mathbf{l}$ can be done with the decomposition of $\mathbf{X}_{\text{ks}}$:

$$\mathbf{R}_{\text{ks}}^{\text{T}}\mathbf{a} = \mathbf{f}^{\text{T}}, \tag{B.24a}$$

$$\mathbf{R}_{\text{ks}}\mathbf{l} = \mathbf{a}. \tag{B.24b}$$

As $\mathbf{R}_{\text{ks}}$ is upper triangular, the calculations are swift. With $\mathbf{l}$, the update of the decomposition $\mathbf{P}^{-1}$ due to (B.23b), is done by a *Cholesky update* and can be found in literature on linear algebra, e.g. (Golub and Van Loan, 1996; Stewart, 1998).

B.2.2 Adding a new key sample

The second update treated is the addition of a key sample into the set of key samples. This update is given in algorithm 4.1(b). The indicator matrix $\mathbf{X}$ is extended by a row and a column due to the inclusion of a key sample, and the decomposition should be updated accordingly. Although the update can be done by reasoning from the QR decomposition, the approach used, uses the Cholesky decomposition.

After the update the decomposition should still fulfill:

$$\bar{\mathbf{R}}_{\text{a}}^{\text{T}}\bar{\mathbf{R}}_{\text{a}} = \bar{\mathbf{X}}_{\text{ks,a}}^{\text{T}}\bar{\mathbf{V}}^{-1}\bar{\mathbf{X}}_{\text{ks,a}}, \tag{B.25}$$

in which

$$\bar{\mathbf{X}}_{\text{ks,a}} = \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} & \mathbf{y} \\ \mathbf{f} & \phi & y \end{bmatrix} \tag{B.26}$$

denotes the augmented matrix containing the key sample indicator matrix. $\bar{\mathbf{R}}_{\text{a}}^{\text{T}}$ is as in (B.15), $\bar{\mathbf{V}}^{-1}$ is the updated weight matrix. Because the assumption that the new key sample is uncorrelated with the present key samples, this matrix is extended with a row and a column and the lower right element becomes 1. Substitution of the partitioned matrices in (B.25) yields:

$$\begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} & \mathbf{y} \\ \mathbf{f} & \phi & y \end{bmatrix}^{\text{T}} \begin{bmatrix} \mathbf{V}^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f}^{\text{T}} & \mathbf{y} \\ \mathbf{f} & \phi & y \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{R}} & \mathbf{g} & \bar{\mathbf{r}} \\ 0 & \gamma & \bar{\rho} \\ 0 & 0 & \tau \end{bmatrix}^{\text{T}} \begin{bmatrix} \bar{\mathbf{R}} & \mathbf{g} & \bar{\mathbf{r}} \\ 0 & \gamma & \bar{\rho} \\ 0 & 0 & \tau \end{bmatrix}, \tag{B.27}$$

which gives, after multiplication:

$$\begin{bmatrix} \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}} + \mathbf{f}^{\text{T}}\mathbf{f} & \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{f}^{\text{T}} + \mathbf{f}^{\text{T}}\phi & \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{y} + \mathbf{f}^{\text{T}}y \\ \mathbf{f}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}} + \phi\mathbf{f} & \mathbf{f}\mathbf{V}^{-1}\mathbf{f}^{\text{T}} + \phi^2 & \mathbf{f}\mathbf{V}^{-1}\mathbf{y} + \phi y \\ \mathbf{y}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}} + y\mathbf{f} & \mathbf{y}^{\text{T}}\mathbf{V}^{-1}\mathbf{f}^{\text{T}} + \phi t & \mathbf{y}^{\text{T}}\mathbf{V}^{-1}\mathbf{y} + y^2 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{R}}^{\text{T}}\bar{\mathbf{R}} & \bar{\mathbf{R}}^{\text{T}}\mathbf{g} & \bar{\mathbf{R}}^{\text{T}}\bar{\mathbf{r}} \\ \mathbf{g}^{\text{T}}\bar{\mathbf{R}} & \mathbf{g}^{\text{T}}\mathbf{g} + \gamma^2 & \mathbf{g}^{\text{T}}\bar{\mathbf{r}} + \gamma\bar{\rho} \\ \bar{\mathbf{r}}^{\text{T}}\bar{\mathbf{R}} & \bar{\mathbf{r}}^{\text{T}}\mathbf{g} + \gamma\bar{\rho} & \bar{\mathbf{r}}^{\text{T}}\bar{\mathbf{r}} + \bar{\rho}^2 + \tau^2 \end{bmatrix}. \tag{B.28}$$

This gives all the necessary equations for the update. However, it is possible to first use the training sample to fine tune the key samples, and second, include the extra key sample. This is advantageous, because the ς^2, found with the fine tuning of the key samples, can be used to test whether the sample needs to be included for a key sample; the inclusion of the information needs to be done in both situations.

To show that first the row can be added and then the column, we show the update without the extra column.

$$\begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{y} \\ \mathbf{f} & y \end{bmatrix}^{\text{T}} \begin{bmatrix} \mathbf{V}^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{y} \\ \mathbf{f} & y \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{R}} & \bar{\mathbf{r}} \\ 0 & \bar{\rho} \end{bmatrix}^{\text{T}} \begin{bmatrix} \bar{\mathbf{R}} & \bar{\mathbf{r}} \\ 0 & \bar{\rho} \end{bmatrix}, \tag{B.29a}$$

$$\begin{bmatrix} \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}} + \mathbf{f}^{\text{T}}\mathbf{f} & \mathbf{X}_{\text{ks}}^{\text{T}}\mathbf{V}^{-1}\mathbf{y} + \mathbf{f}^{\text{T}}y \\ \mathbf{y}^{\text{T}}\mathbf{V}^{-1}\mathbf{X}_{\text{ks}} + y\mathbf{f} & \mathbf{y}^{\text{T}}\mathbf{V}^{-1}\mathbf{y} + y^2 \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{R}}^{\text{T}}\bar{\mathbf{R}} & \bar{\mathbf{R}}^{\text{T}}\bar{\mathbf{r}} \\ \bar{\mathbf{r}}^{\text{T}}\bar{\mathbf{R}} & \bar{\mathbf{r}}^{\text{T}}\bar{\mathbf{r}} + \bar{\rho}^2 \end{bmatrix}. \tag{B.29b}$$

It can be seen that calculation of $\bar{\mathbf{R}}$ and $\bar{\mathbf{r}}$ are the same in (B.28) and (B.29), and therefore the fine tuning of the key sample can be done first.

$\bar{\mathbf{r}}$ equals the updated $\mathbf{z}_{\mathbf{X}}$. The remaining parts of the new decomposition can be calculated as:

$$\bar{\mathbf{R}}^{\mathrm{T}}\mathbf{g} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \mathbf{f}^{\mathrm{T}}\phi, \tag{B.30a}$$

$$\gamma^2 = \mathbf{f}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \phi^2 - \mathbf{g}^{\mathrm{T}}\mathbf{g}, \tag{B.30b}$$

$$\bar{\rho} = \frac{1}{\gamma}\left(\mathbf{f}\mathbf{V}^{-1}\mathbf{y} + \phi y - \mathbf{g}^{\mathrm{T}}\bar{\mathbf{r}}\right). \tag{B.30c}$$

$\mathbf{P}^{-1}$ is trivially updated as:

$$\bar{\mathbf{P}}^{-1} = \begin{bmatrix} \mathbf{P}^{-1} & 0 \\ 0 & 1 \end{bmatrix}. \tag{B.31}$$

As the key samples alter, the decomposition $\mathbf{R}_{\mathrm{ks}}$ has to be updated too. With the help of theorem 4 this can be readily done. The situation before the update is:

$$\mathbf{R}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{R}_{\mathrm{ks}} = \mathbf{X}_{\mathrm{ks}}, \tag{B.32}$$

while after the update

$$\begin{bmatrix} \mathbf{R}_{\mathrm{ks}} & \mathbf{r} \\ 0 & \rho \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{R}_{\mathrm{ks}} & \mathbf{r} \\ 0 & \rho \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{\mathrm{ks}} & \mathbf{f}^{\mathrm{T}} \\ \mathbf{f} & \phi \end{bmatrix}, \tag{B.33a}$$

$$\begin{bmatrix} \mathbf{R}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{R}_{\mathrm{ks}} & \mathbf{R}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{r} \\ \mathbf{r}^{\mathrm{T}}\mathbf{R}_{\mathrm{ks}} & \mathbf{r}^{\mathrm{T}}\mathbf{r} + \rho^2 \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{\mathrm{ks}} & \mathbf{f}^{\mathrm{T}} \\ \mathbf{f} & \phi \end{bmatrix} \tag{B.33b}$$

has to hold. So, the $\mathbf{R}_{\mathrm{ks}}$ does not change, $\mathbf{r} = \mathbf{R}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{f}^{\mathrm{T}}$, while $\rho^2 = \phi - \mathbf{r}^{\mathrm{T}}\mathbf{r}$. ρ can be used to test if the new key sample contains enough information.

B.2.3 Removing a key sample

The last update is the omission of a key sample and is given in algorithm 4.1(c). The implementation for the omission of a key sample consists of two steps:

1) Move the induced vector corresponding to the key sample to the right-most position of the indicator matrix.
2) Remove this column from the matrix.

The change in the decomposition due to the switching of columns in the indicator matrix can be calculated by partitioning the augmented decomposition matrix:

$$\mathbf{R}_{\mathrm{a}} = \left[\mathbf{R}_1 \quad \mathbf{p} \quad \mathbf{R}_2 \,\middle|\, \mathbf{z}_{\mathbf{X}}\right]. \tag{B.34}$$

In this, $\mathbf{p}$ is the column corresponding to the column in the indicator matrix that is omitted. The columns of $\mathbf{R}_a$ are changed to:

$$\left[\mathbf{R}_1 \quad \mathbf{R}_2 \quad \mathbf{p} \,\middle|\, \mathbf{z}_\chi\right]. \tag{B.35}$$

By means of Givens rotations, the left part can be made upper triangular, which is the updated decomposition:

$$\bar{\mathbf{R}}_a = \left[\bar{\mathbf{R}}_1 \quad \bar{\mathbf{R}}_2 \quad \bar{\mathbf{p}} \,\middle|\, \bar{\mathbf{z}}_\chi\right]. \tag{B.36}$$

Details concerning the changing of the columns in the decomposition are again found in (Golub and Van Loan, 1996; Stewart, 1998).

The last element of $\mathbf{z}_\chi$ shows the (weighted) error increase due to the removal of this indicator. If this error is larger than specified in the removal criterion, see section 4.2.3, the omission should not be done.

The removal of the last column of the indicator matrix form the decomposition is readily implemented. This column can just be omitted from the decomposition.

The update of the decomposition of $\mathbf{V}^{-1}$ contains the same two steps as for the update of $\mathbf{R}$. The changing of the location of the columns is first performed, while after this, the column and row can be removed. From algorithm 4.1(c) it is found that the matrix $\mathbf{V}^{-1}$ is altered by the omission of the last row/column. This change needs to be done for the decomposition too. Before the update, the matrix $\mathbf{V}^{-1}$ is partitioned as:

$$\mathbf{V}^{-1} = \begin{bmatrix} \mathbf{W}^{-1} & \mathbf{v} \\ \mathbf{v}^T & \nu^{-1} \end{bmatrix}. \tag{B.37}$$

The last row and column are removed due to the omission of the key sample, and the updated weight matrix becomes:

$$\bar{\mathbf{V}}^{-1} = \mathbf{W}^{-1} + \mathbf{l}\mathbf{v}^T + \mathbf{v}\mathbf{l}^T + \mathbf{l}\nu^{-1}\mathbf{l}^T. \tag{B.38}$$

The following two vectors are constructed:

$$\mathbf{a} = \tfrac{1}{2}\mathbf{l} + \tfrac{1}{2}\mathbf{v}, \tag{B.39a}$$

$$\mathbf{b} = \tfrac{1}{2}\mathbf{l} - \tfrac{1}{2}\mathbf{v}. \tag{B.39b}$$

With these, (B.38) alters to:

$$\bar{\mathbf{V}}^{-1} = \mathbf{W}^{-1} + \mathbf{a}\mathbf{a}^T - \mathbf{b}\mathbf{b}^T + \mathbf{l}\nu^{-1}\mathbf{l}^T. \tag{B.40}$$

In order to update the Cholesky decomposition $\mathbf{P}^{-1}$ of $\mathbf{V}^{-1}$, two rank 1 Cholesky updates are required, and one rank 1 *downdate*. These updates are available in the literature.

B.2.4 Calculation of the weight

In the previous sections, the updates for the decomposition of the weight were treated. The weight matrix, or its decomposition, can also be calculated from $\mathbf{R}_{ks}$ and $\mathbf{R}$. The weight is determined as follows (4.19):

$$\mathbf{V}^{-1} = \mathbf{X}_{ks}^{-T}\mathbf{X}^{T}\mathbf{X}\mathbf{X}_{ks}^{-1}. \tag{B.41}$$

Using the decompositions of table B.1:

$$\mathbf{V}^{-1} = \mathbf{R}_{ks}^{T}\mathbf{R}_{ks}^{-T}\mathbf{R}^{T}\mathbf{R}\mathbf{R}_{ks}^{-T}\mathbf{R}_{ks}^{T}. \tag{B.42}$$

This can be calculated by:

$$\mathbf{P}^{-1}\mathbf{R}_{ks}\mathbf{R}_{ks}^{T} = \mathbf{R}, \tag{B.43a}$$

$$\mathbf{V}^{-1} = \mathbf{P}^{-T}\mathbf{P}^{-1}. \tag{B.43b}$$

Appendix C

Conditions on new weight matrix

FOR THE INCLUSION of a new key sample an assumption is required concerning the correlation between the key samples, because the actual correlation cannot be calculated due to the omission of the old samples. In section 4.2.2 it was shown that the following matrix equality has to hold after the addition of a key sample:

$$\begin{bmatrix} \mathbf{X} & \mathbf{z} \\ \mathbf{f} & \phi \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{X} & \mathbf{z} \\ \mathbf{f} & \phi \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{\mathrm{ks}} & \mathbf{f}^{\mathrm{T}} \\ \mathbf{f} & \phi \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{V}^{-1} & \mathbf{v}^{\mathrm{T}} \\ \mathbf{v} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\mathrm{ks}} & \mathbf{f}^{\mathrm{T}} \\ \mathbf{f} & \phi \end{bmatrix}. \tag{C.1}$$

As $\mathbf{z}$ is the vector with the values of the new indicator vector with *all* the training samples, this vector cannot be calculated. This results in the unknown vector $\mathbf{v}$ that indicates the correlation between the key samples.

As the actual correlation between the key samples cannot be determined, we can use the vector $\mathbf{v}$ to impose a correlation between the key samples. This might be used to implement some kind of spatial filtering. It should be known however, whether the selected $\mathbf{v}$ gives rise to a real-valued vector $\mathbf{z}$. Throughout this thesis, $\mathbf{v}$ was set at 0. It is shown in this appendix that this choice always result in a real-valued vector $\mathbf{z}$.

Equating the elements of the matrices of (4.41), yield the following three matrix equations:

$$\mathbf{X}^{\mathrm{T}}\mathbf{X} + \mathbf{f}^{\mathrm{T}}\mathbf{f} = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{X}_{\mathrm{ks}} + \mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{X}_{\mathrm{ks}} + \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{v}\mathbf{f} + \mathbf{f}^{\mathrm{T}}\mathbf{f}, \tag{C.2a}$$

$$\mathbf{X}^{\mathrm{T}}\mathbf{z} + \mathbf{f}^{\mathrm{T}}\phi = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{f}^{\mathrm{T}} + \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{v}\phi + \mathbf{f}^{\mathrm{T}}\phi, \tag{C.2b}$$

$$\mathbf{z}^{\mathrm{T}}\mathbf{z} + \phi^2 = \mathbf{f}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \phi\mathbf{v}^{\mathrm{T}}\mathbf{f}^{\mathrm{T}} + \mathbf{f}\mathbf{v}\phi + \phi^2. \tag{C.2c}$$

The last two of these equations yield:

$$\mathbf{X}^{\mathrm{T}}\mathbf{z} = {\mathbf{X}_{\mathrm{ks}}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \mathbf{f}^{\mathrm{T}}\mathbf{v}^{\mathrm{T}}\mathbf{f}^{\mathrm{T}} + \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{v}\phi = \mathbf{b}, \tag{C.3a}$$

$$\mathbf{z}^{\mathrm{T}}\mathbf{z} = \mathbf{f}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + 2\phi\mathbf{v}^{\mathrm{T}}\mathbf{f}^{\mathrm{T}} = c. \tag{C.3b}$$

In which the right-hand side of these equations are known. The first matrix equation consists of N linear equations with k unknowns, $k \leq N$. As a result, $\mathbf{z}$ is not uniquely determined by this set of equations, and an $(N-k)$-dimensional null-space exists. This null-space can be used to fulfil (C.3b). If a solution to the second equation exists, then there is a possible real-valued vector $\mathbf{z}$. This solution is found by means of a QR decomposition. Starting from:

$$\mathbf{X}^{\mathrm{T}}\mathbf{z} = \mathbf{b}, \tag{C.4a}$$

$$\mathbf{z}^{\mathrm{T}}\mathbf{z} = c. \tag{C.4b}$$

We can use the QR decomposition of $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix}\mathbf{Q}_{\mathbf{X}} & \mathbf{Q}_{\perp}\end{bmatrix}\begin{bmatrix}\mathbf{R} \\ \mathbf{0}\end{bmatrix}, \tag{C.5}$$

$$\mathbf{X}^{\mathrm{T}} = \begin{bmatrix}\mathbf{R}^{\mathrm{T}} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{Q}_{\mathbf{X}}^{\mathrm{T}} \\ \mathbf{Q}_{\perp}^{\mathrm{T}}\end{bmatrix}. \tag{C.6}$$

In this, $\mathbf{Q}_{\mathbf{X}}$ spans the same space as the matrix $\mathbf{X}$ and $\mathbf{Q}_{\perp}$ spans the null-space of $\mathbf{X}$. The matrix $\mathbf{R}$ can be calculated from the Cholesky decomposition of $\mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{X}_{\mathrm{ks}}$ and is therefore known. Defining the vector:

$$\hat{\mathbf{z}} = \mathbf{Q}^{\mathrm{T}}\mathbf{z} = \begin{bmatrix}\mathbf{Q}_{\mathbf{X}}\mathbf{z} \\ \mathbf{Q}_{\perp}\mathbf{z}\end{bmatrix} = \begin{bmatrix}\hat{\mathbf{z}}_1 \\ \hat{\mathbf{z}}_2\end{bmatrix}. \tag{C.7}$$

The size of $\hat{\mathbf{z}}_1$ is k while $\hat{\mathbf{z}}_2$ is $N-k$. Now we can solve:

$$\mathbf{X}\mathbf{z} = \mathbf{b}, \tag{C.8}$$

$$\begin{bmatrix}\mathbf{R}^{\mathrm{T}} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{Q}_{\mathbf{X}}^{\mathrm{T}} \\ \mathbf{Q}_{\perp}^{\mathrm{T}}\end{bmatrix}\mathbf{z} = \mathbf{b}, \tag{C.9}$$

$$\begin{bmatrix}\mathbf{R}^{\mathrm{T}} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{Q}_{\mathbf{X}}^{\mathrm{T}}\mathbf{z} \\ \mathbf{Q}_{\perp}^{\mathrm{T}}\mathbf{z}\end{bmatrix} = \mathbf{b}, \tag{C.10}$$

$$\begin{bmatrix}\mathbf{R}^{\mathrm{T}} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\hat{\mathbf{z}}_1 \\ \hat{\mathbf{z}}_2\end{bmatrix} = \mathbf{b}, \tag{C.11}$$

$$\mathbf{R}^{\mathrm{T}}\hat{\mathbf{z}}_1 = \mathbf{b}. \tag{C.12}$$

So, the partial solution is given in terms of $\hat{\mathbf{z}}_1$ and $\hat{\mathbf{z}}_2$ is free. With this the second equation can be fulfilled. Note that:

$$\mathbf{z}^{\mathrm{T}}\mathbf{z} = \mathbf{z}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{Q}\mathbf{z} = \hat{\mathbf{z}}^{\mathrm{T}}\hat{\mathbf{z}} = \hat{\mathbf{z}}_1^{\mathrm{T}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2^{\mathrm{T}}\hat{\mathbf{z}}_2. \tag{C.13}$$

Three situations can occur:

1) $\hat{\mathbf{z}}_1^{\mathrm{T}}\hat{\mathbf{z}}_1 > c$. $\hat{\mathbf{z}}_2$ has to be complex. This means that for the chosen value of $\mathbf{v}$, no real indicator vector can be found that represents this weight.

2) $\hat{\mathbf{z}}_1^{\mathrm{T}}\hat{\mathbf{z}}_1 = c$. In this case there is a unique solution $\hat{\mathbf{z}}_2 = 0$. There is one solution for $\mathbf{z}$.

3) $\hat{\mathbf{z}}_1^{\mathrm{T}}\hat{\mathbf{z}}_1 < c$. In this case the vector $\hat{\mathbf{z}}_2$ and therefore $\mathbf{z}$, are not uniquely defined. However, there are solutions to the problem which means that the weight does represent some set of possible values for the indicators, but these are not uniquely defined

The sample that fulfils the condition is indicated as an 'imaginary sample.' This sample might never be presented to the approximator, or even equals an indicator vector that a normal training sample can induce.

Throughout the thesis, we assumed that the key sample newly included was uncorrelated with the already present key samples, i.e. $\mathbf{v} = 0$. We want to calculated if the choice of $\mathbf{v} = 0$ always results in a real indicator vector. Inserting $\mathbf{v} = 0$ in (C.3) gives:

$$\mathbf{X}^{\mathrm{T}}\mathbf{z} = {\mathbf{X}_{\mathrm{ks}}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}}, \tag{C.14a}$$
$$\mathbf{z}^{\mathrm{T}}\mathbf{z} = \mathbf{f}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}}. \tag{C.14b}$$

Using (C.12) gives:

$$\mathbf{R}^{\mathrm{T}}\hat{\mathbf{z}}_1 = \mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}}. \tag{C.15}$$

Fulfilling the second condition:

$$\begin{aligned}
\mathbf{z}^{\mathrm{T}}\mathbf{z} &= \hat{\mathbf{z}}_1^{\mathrm{T}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2^{\mathrm{T}}\hat{\mathbf{z}}_2, && \text{(C.16a)}\\
&= \mathbf{f}\mathbf{V}^{-\mathrm{T}}\mathbf{X}_{\mathrm{ks}}\mathbf{R}^{-1}\mathbf{R}^{-T}\mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \hat{\mathbf{z}}_2^{\mathrm{T}}\hat{\mathbf{z}}_2, && \text{(C.16b)}\\
&= \mathbf{f}\mathbf{V}^{-\mathrm{T}}\mathbf{X}_{\mathrm{ks}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})\mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \hat{\mathbf{z}}_2^{\mathrm{T}}\hat{\mathbf{z}}_2, && \text{(C.16c)}\\
&= \mathbf{f}\mathbf{V}^{-\mathrm{T}}\mathbf{X}_{\mathrm{ks}}(\mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{X}_{\mathrm{ks}})\mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \hat{\mathbf{z}}_2^{\mathrm{T}}\hat{\mathbf{z}}_2, && \text{(C.16d)}\\
&= \mathbf{f}\mathbf{V}^{-\mathrm{T}}\mathbf{X}_{\mathrm{ks}}\mathbf{X}_{\mathrm{ks}}^{-1}\mathbf{V}\mathbf{X}_{\mathrm{ks}}^{-\mathrm{T}}\mathbf{X}_{\mathrm{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} + \hat{\mathbf{z}}_2^{\mathrm{T}}\hat{\mathbf{z}}_2, && \text{(C.16e)}\\
&= \mathbf{f}\mathbf{V}^{-1}\mathbf{f} + \hat{\mathbf{z}}_2^{\mathrm{T}}\hat{\mathbf{z}}_2, && \text{(C.16f)}
\end{aligned}$$

requires that $\hat{\mathbf{z}}_2 = 0$. Therefore, there is always a unique solution if the correlations are set to zero.

Example C.1 (Imaginary sample)
An example might give some insight. The dual indicator function that is used is $k(x, x_i) = 1 + \min(x, x_i)$. The training data and the key samples as well as the corresponding indicator matrices are given as:

$$(x,y)_i = \begin{bmatrix} 0 & 1 \\ 0 & 2 \\ 1 & 3 \\ 2 & 5 \\ 3 & 7 \end{bmatrix}, \quad (x_{\text{ks}})_i = \begin{bmatrix} 0 \\ 3 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}, \quad \mathbf{X}_{\text{ks}} = \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix}. \tag{C.17}$$

The resulting weight matrix can be calculated from these:

$$\mathbf{V}^{-1} = \begin{bmatrix} 2.56 & 0.44 \\ 0.44 & 1.56 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 2.23 & 4.92 \\ 0 & 2.61 \end{bmatrix}. \tag{C.18}$$

A new sample is introduced $(x, y) = (1.5, 4)$. This sample is added to the set of present key samples. With the given kernel function the resulting indicator vector becomes $\mathbf{f} = \begin{bmatrix} 1 & 2.5 \end{bmatrix}$. The innerproduct in the feature space of this sample with itself is $\phi = k(1.5, 1.5) = 2.5$. With these variables, the value of $\hat{\mathbf{z}}_1$ can be calculated (C.12):

$$\mathbf{X}^{\mathrm{T}}\mathbf{z} = \mathbf{X}_{\text{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} = \mathbf{b} = \begin{bmatrix} 8 \\ 21 \end{bmatrix}, \tag{C.19}$$

$$\hat{\mathbf{z}}_1 = \mathbf{R}^{-\mathrm{T}}\mathbf{b} = \begin{bmatrix} 3.577 \\ 1.304 \end{bmatrix}. \tag{C.20}$$

The second condition (C.3b) states that $\mathbf{z}^{\mathrm{T}}\mathbf{z} = \mathbf{f}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}}$:

$$\mathbf{f}\mathbf{V}^{-1}\mathbf{f}^{\mathrm{T}} = 14.5, \tag{C.21}$$

$$\hat{\mathbf{z}}_1^{\mathrm{T}}\hat{\mathbf{z}}_1 = 14.5. \tag{C.22}$$

From which it follows that $\hat{\mathbf{z}}_2$ equals zero and the solution for $\mathbf{z}$ is unique.

Because we know $\mathbf{X}$ in this example, we can calculate $\mathbf{z}$. This is normally impossible.

$$\mathbf{Q}\hat{\mathbf{z}}_1 = \begin{bmatrix} \frac{1}{\sqrt{5}}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \frac{1}{\sqrt{6.8}}\begin{bmatrix} -1.2 \\ -1.2 \\ 0.2 \\ 0.8 \\ 1.8 \end{bmatrix} \end{bmatrix} \begin{bmatrix} 3.577 \\ 1.3038 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1.5 \\ 2 \\ 2.5 \end{bmatrix}. \tag{C.23}$$

This indicator vector cannot be induced by a training sample with the old training samples. The values for this indicator vector with the old training samples

are:

$$\mathbf{z}_{\text{true}} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2.5 \\ 2.5 \end{bmatrix}. \tag{C.24}$$

▩

Next to the update of the weight matrix, the target for the new key sample has to be given a value. The new value can be chosen freely. Before the update the following holds:

$$\mathbf{X}^{\mathrm{T}}\mathbf{y} = \mathbf{X}_{\text{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{y}_{\text{ks}}. \tag{C.25}$$

while after the update the following has to hold:

$$\begin{bmatrix} \mathbf{X} & \mathbf{z} \\ \mathbf{f} & \phi \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \mathbf{y} \\ y \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{\text{ks}} & \mathbf{f} \\ \mathbf{f}^{\mathrm{T}} & \phi \end{bmatrix} \begin{bmatrix} \mathbf{V}^{-1} & \mathbf{v} \\ \mathbf{v}^{\mathrm{T}} & \phi \end{bmatrix} \begin{bmatrix} \mathbf{y}_{\text{ks}} \\ y_{\text{ks}} \end{bmatrix}. \tag{C.26}$$

Setting the left-hand side equal to the right-hand side results in the following two equation for $\mathbf{v} = 0$:

$$\mathbf{X}^{\mathrm{T}}\mathbf{y} + \mathbf{f}^{\mathrm{T}}y = \mathbf{X}_{\text{ks}}^{\mathrm{T}}\mathbf{V}^{-1}\mathbf{y}_{\text{ks}} + \mathbf{f}^{\mathrm{T}}y_{\text{ks}}, \tag{C.27a}$$

$$\mathbf{z}^{\mathrm{T}}\mathbf{y} + \phi y = \mathbf{f}\mathbf{V}^{-1}\mathbf{y}_{\text{ks}} + \phi y_{\text{ks}}. \tag{C.27b}$$

The first of these equations gives $y = y_{\text{ks}}$. If we take an arbitrary value for y_{ks}, this will give a possible situation for the full data with real values of $\mathbf{y}$. This is because the product of $\mathbf{z}^{\mathrm{T}}\mathbf{y}$ can be set to any value required so (C.27b) can always be fulfilled. The new y_{ks} can therefore be chosen freely.

Bibliography

Aha, D. W. "Editorial." *Artificial Intelligence Review*, vol. 11, no. 1/5, pp. 1–6, 1997.

Aitken, A. C. "On least squares and linear combinations of observations." *Proceedings of the Royal Society of Edinburgh*, vol. 55, pp. 42–47, 1934.

Aoki, M. *Introduction to Optimization Techniques, Fundamentals and Applications of Nonlinear Programming*. Macmillan Series in Applied Computer Sciences. The Macmillan Company, New York, 1971.

Arimoto, S., S. Kawamura, and F. Miyazaki. "Bettering operations of robots by learning." *Journal of Robotic Systems*, vol. 1, no. 2, pp. 123–140, 1984.

Åström, K. J., and K. Wittenmark. *Computer- Controlled Systems: Theory and Design*. Prentice Hall Information and System Sciences Series. Prentice Hall, Inc., Upper Saddle River, New Jersey, 3rd edn., 1997. ISBN 0-13-314899-8.

Atkenson, G. C., A. W. Moore, and S. Schaal. "Locally weighted learning." *Artificial Intelligence Review*, vol. 11, no. 1/5, pp. 11–73, 1997.

Björck, Å. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1996.

Boyd, S., and L. Vandenberghe. *Convex Optimization*. Cambridge university press, Cambridge, UK, 2004.

Brown, M., and C. Harris. *Neurofuzzy adaptive modeling and control*. Prentice Hall International, UK, 1994. ISBN 0-13-134453-6.

Campbell, C. "An introduction to kernel methods." In R. J. Howlett, and L. C. Jain, eds., *Radial Basis Function Networks: Design and Applications*, p. 31. Springer Verlag, Berlin, 2000.

Campbell, C. "Kernel methods: A survey of current techniques." *Neurocomputing*, vol. 48, pp. 63 – 84, 2002.

Cao, L. J., K. S. Chua, W. K. Chong, H. P. Lee, and Q. M. Gu. "A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine." *Neurocomputing*, vol. 55, pp. 321 – 336, 2003.

Chan, W. C., C. W. Chan, K. C. Cheung, and C. J. Harris. "On the modelling of nonlinear dynamic systems using support vector neural networks." *Engineering Applications of Artificial Intelligence*, vol. 14, pp. 105–113, 2001.

Chang, C.-C., and C.-J. Lin. "LIBSVM: A library for support vector machines." Tech. rep., Department of Computer Science and Information Engineering, National Taiwan University, 2003. Available from: http://www.csie.ntu.edu.tw/˜cjlin/libsvm/.

Chen, S. "Nonlinear time series modelling and prediction using gaussian RBF networks with enhanced clustering and RLS learning." *Electronics Letters*, vol. 31, no. 2, pp. 117–118, 1995.

Chen, S., F. N. Cowan, and P. M. Grant. "Orthogonal least squares learning algorithm for radial basis function networks." *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302–309, 1991.

Conradt, J., G. Tevatia, S. Vijayakumar, and S. Schaal. "On-line learning for humanoid robot systems." In *Proc. Of Seventeenth International Conference on Machine Learning (ICML2000)*, pp. 191–198. Stanford, 2000.

Cristianini, N., and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-base learning methods*. Cambridge University Press, Cambridge, UK, 2000.

Cun, Y. L., J. S. Denker, and S. A. Solla. "Optimal brain damage." In Touretzky, ed., *Advances in Neural Information Processing Systems*, pp. 1481–1497. Morgen Kaufmann, San Mateo, CA, 1990.

De Kruif, B. J., and T. J. A. de Vries. "Motion control using support vector machine-based learning feed-forward." In *Proceedings WESIC 2001*, pp. 169–178. University of Twente, Enschede, The Netherlands, 2001a. ISBN 90-365-16102.

De Kruif, B. J., and T. J. A. de Vries. "On using a support vector machine in learning feed-forward control." In *Proc. Int. Conf. On Advanced Intelligent Mechatronics (AIM'01)*, pp. 272–277. IEEE/ASME, Como,Italy, 2001b.

De Kruif, B. J., and T. J. A. de Vries. "Improving price/performance ratio of a linear motor by means of learning control." In *Proceedings of the IFAC Mechatronics 2002*. Berkeley, USA, 2002a.

De Kruif, B. J., and T. J. A. de Vries. "Support-vector-based least squares for learning non-linear dynamics." In *Proceedings of the CDC'02*. 2002b.

De Kruif, B. J., and T. J. A. de Vries. "On-line nonparametric regression to learn state- dependent disturbances." In *Proceedings of the 18th IEEE International Symposium on Intelligent Control (ISIC'03)*, pp. 75–80. Omnipress, Houston, USA, 2003a.

De Kruif, B. J., and T. J. A. de Vries. "Phase correction for learning feedforward control." In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM'03)*. Kobe, Japan, 2003b.

De Kruif, B. J., and T. J. A. de Vries. "Pruning error minimization in least squares support vector machines." *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 696 – 702, 2003c.

De Kruif, B. J., and T. J. A. de Vries. "Comparison of four support-vector based function approximators." In *Proceedings of the IJCNN'04*. Boedapest, 2004a. Invited session, accepted.

De Kruif, B. J., and T. J. A. de Vries. "Evaluation of four sample based function approximators." *Automatica*, 2004b. Submitted to.

De Kruif, B. J., B. van Wermeskerken, T. J. A. de Vries, and M. J. Korsten. "Sensor fusion for linear motors, an approach for low-cost measurements." In *8th Mechatronics Forum International Conference*, pp. 554–563. Drebbel Insitute for Mechatronics, University of Twente, The Netherlands, 2002.

De Roover, D., and O. H. Bosgra. "Synthesis of robust multivariable iterative learning controllers with application to a wafer stage motion system." *International Journal of Control*, vol. 73, no. 10, pp. 968–979, 2000. Special issue on Iterative Learning Control.

De Vries, T. J. A., W. J. R. Velthuis, and L. J. Idema. "Application of parsimonious learning feedforward control to mechatronic systems." *IEE Proc. D: Control Theory & Applications*, vol. 148, no. 4, pp. 318–322, 2001.

Draper, N. R., and H. Smith. *Applied Regression Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., New York, 3rd edn., 1998.

Eglence, M. *Design and Realization of a Safe Control System for a Parallel Manipulator*. Master's thesis, University of Twente, 2003. Nr. 010CE2003.

Esposito, A., M. Marinaro, and S. Scarpetta. "An incremental local radial basis function network." In *ESANN'1998 Proceedings - European Symposium on Artificial Neural Networks Bruges (Belgium)*, pp. 21–26. D-Facto public., 1998.

Fahlman, S. E., and C. Lebiere. "The cascade-correlation learning architecture." In D. S. Touretzky, ed., *Advances in Neural Information Processing Systems*, vol. 2, pp. 524–532. Morgan Kaufmann, San Mateo, Denver 1989, 1990.

Fisher, R. A. "On an absolute criterion for fitting frequency curves." *Messenger of Mathematics*, vol. 41, pp. 155–160, 1912.

Franklin, G. F., J. D. Powel, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Addison-Wesley Publishing Company, Reading, Massachusetts, 3rd edn., 1994.

Fun, M. H., and M. T. Hagan. "Recursive orthogonal least squares learning with automatic weight selection for gaussian neural networks." In *International Joint Conference on Neural Networks*. Washington, 1999.

Funival, G. M., and R. W. Wildon Jr. "Regressions by leaps and bounds." *Technometrics*, vol. 16, no. 4, pp. 499–511, 1974.

Furman, B., D. Pinkernell, and S. Elgee. "Case studies on design of mechatronic products." *IEEE Transactions on components, packaging, and manufacturing technology – Part C*, vol. 20, no. 1, pp. 8–13, 1997.

Gestel, T. V., J. Suykens, B. de Moor, and J. Vandewalle. "Automatic relevance determination for least squares support vector machines regression." In *International Joint Conference on Neural Networks, IJCNN'01*. 2001.

Ghosh, J., and A. Nag. *An Overview of Radial Basis Function Networks*, chap. 1, pp. 1–30. Physica-Verlag, New York, 2001.

Gieras, J. F., and Z. J. Piech. *Linear Synchronous Motors, Transportation and Automation Systems*. CRC Press, Boca Raton, Florida, 2000. ISBN 0-8493-1859-7.

Girosi, F., M. Jones, and T. Poggio. "Priors stabilizers and basis functions: From regularization to radial, tensor and additive splines." Tech. Rep. AIM-1430, 1993.

Golub, G. H., and C. Van Loan. *Matrix computations*. The Johns Hopkins University Press, Baltimore, Maryland, 3rd edn., 1996.

Gomm, J. B. "Selecting radial basis function network centers with recursive orthogonal least squares training." *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 306–314, 2000.

Gorinevsky, D., and G. Vukovich. "Model-based update in task-level feedforward control using on-line approximation." *Automatica*, vol. 37, pp. 391–400, 2001.

Gunnarsson, S., and M. Norrlöf. "On the design of ILC algorithms using optimization." *Automatica*, vol. 37, pp. 2011–2016, 2001.

Hall, P. "On projection pursuit regression." *Annals of Statistics*, vol. 17, no. 2, pp. 573–588, 1989.

Han, S.-H., Y.-H. Kim, and I.-J. Ha. "Iterative identification of state-dependent disturbance torque for high-precision velocity control of servo motors." *IEEE Trans. on Automatic Control*, vol. 43, no. 5, pp. 724–729, 1998.

Hassibi, B., and D. G. Stork. "Second order derivatives for network pruning: Optimal brain surgeon." In S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., *Advances in Neural Information Processing Systems*, vol. 5, pp. 164–171. Morgan Kaufmann, San Mateo, CA, 1993.

Haykin, S. *Neural Networks, A Comprehensive Foundation*. Prentice Hall, Inc., Upper Saddle River, New Jersey, 1994.

Haykin, S. *Adaptive Filter Theory*. Prentice Hall, Upper Saddle River, New Jersey, 4th edn., 2002.

Hidayat, Z. *Comparison of Learning Methods in the Learning Feed-Forward Control Setting*. Master's thesis, Unversity of Twente, 2004. Nr. 003CE2004.

Hoerl, A. E., and R. W. Kennard. "Ridge regression: Biased estimation for nonorthogonal problems." *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

Horowitz, R. "Learning control applications to mechatronics." *JSME International Journal, Series C*, vol. 37, no. 3, pp. 421–430, 1994. Invited paper.

Huber, P. J. *Robust Statistics*. John Wiley & Sons, Inc., New York, 1981.

Huffel, S. V., and J. Vandewalle. *The Total Least Squares Problem: Computational Aspects and Analysis*. Vol.9 of Fronties in Applied Mathematics. SIAM, Philadelphia, 1991.

Kailath, T. *Linear Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, 1980. ISBN 0-13-536961-4.

Kawato, M., K. Furukawa, and R. Suzuki. "A hierarchical neural network model for control and learning of voluntary movement." *Biological Cybernetics*, vol. 57, pp. 169–185, 1987.

Keerthi, S. S., S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. "Improvements to platt's SMO algorithm for SVM classifier design." *Neural Computation*, vol. 13, pp. 637–649, 2001.

Kleinbaum, D. G., L. L. Kupper, and K. E. Muller. *Applied Regression Analysis and Other Multivariable Methods*. The Duxbury Series in Statistics an Decision Sciences. PWS-KENT Publishing Company, Boston, Massachusetts, 1987.

Leung, C.-S., K.-W. Wong, P.-F. Sum, and L.-W. Chan. "A pruning method for the recursive least squared algorithm." *Neural Networks*, vol. 14, no. 2, pp. 147–174, 2001.

Lippmann, R. P. "Pattern classification using neural networks." *IEEE Communications Magazine*, vol. 27, no. 11, pp. 47–50, 59–64, 1989.

Ljung, L. *System Identification, Theory for the user*. PTR Prentice-Hall information and system sciences series. Prentice-Hall PTR, Upper Saddle River, New Jersey, 2nd edn., 1999. ISBN 0-13-656695-2.

Longman, R. W. *Iterative Learning Control: Analysis, Design, Integration and Applications*, chap. 7: Designing Iterative Learning and Repetitive Controllers, pp. 107–146. Kluwer Academic Publishers, 1998.

Longman, R. W. "Iterative learning control and repetitive control for engineering practice." *International Journal of Control*, vol. 73, no. 10, pp. 930–954, 2000.

Mangasarian, O. L., and D. R. Musicant. "Successive overrelaxation for support vector machines." Tech. Rep. 98-18, University of Wisconsin, Computer Sciences Department, Madison, WI, USA, 1998.

Mercer, J. "Functions of positive and negative type and their connection with the theory of integral equations." *Transactions of the London Philosophical Society*, vol. A, no. 209, pp. 415–446, 1909.

Miller, A. J. *Subset Selection in Regression*. Chapman and Hall, London, 1990.

Moore, K. L. *Iterative Learning Control for Deterministic Systems*. Advances in Industrial Control Series. Springer, Londen, UK, 1992.

Nørgaard, M., O. Ravn, N. K. Poulsen, and L. K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook*. Springer-Verlag, London, 2000.

Otten, G., T. J. A. de Vries, J. van Amerongen, A. M. Rankers, and E. W. Gaal. "Linear motor motion control using a learning feedforward controller." *IEEE/ASME Trans. Mechatronics*, vol. 2, no. 3, pp. 179–187, 1997.

Poggio, T., and F. Girosi. "A theory of networks for approximation and learning." Tech. Rep. A.I. Memo No. 1140, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1989.

Poggio, T., and F. Girosi. "Networks for approximation and learning." *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481–1495, 1990.

Prechelt, L. "Connection pruning with static and adaptive pruning." *Neurocomputing*, vol. 16, pp. 49–61, 1997.

Reed, R. "Pruning algorithms - a survey." *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

Schaal, S., and C. G. Atkeson. "Constructive incremental learning from only local information." *Neural Computation*, vol. 10, no. 8, pp. 2047–2084, 1998.

Schaal, S., C. G. Atkeson, and S. Vijayakumar. "Scalable techniques from nonparameteric statistics for real-time robot learning." *Applied Intelligence - Special issue on Scalable Robotic Applications of Neural Networks*, vol. 1, no. 17, pp. 49–60, 2002.

Schlichthärle, D. *Digital Filters, Basics and Design*. Springer, New York, 2000.

Schölkopf, B., S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola. "Input space vs. feature space in kernel-based methods." *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

Schölkopf, B., P. Y. Simard, A. J. Smola, and V. N. Vapnik. "Prior knowledge in support vector kernels." In M. I. Jordan, M. J. Kearns, and S. A. Solla, eds., *Advances in Neural information processing systems*, vol. 10, pp. 640–646. MIT Press, Cambridge, MA, 1998.

Shevade, S. K., S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. "Improvements to SMO algorithm for SVM regression." Tech. Rep. CD-99-16, National University of Singapore, Dept. of Mechanical and Production Engineering, Control Division, 1999.

Smola, A. J. *Learning with Kernels*. Ph.D. thesis, GMD, Birlinghoven, Germany, 1998.

Smola, A. J., and B. Schölkopf. "A tutorial on support vector regression." Tech. Rep. NC2-TR-1998-030, Royal Holloway, University of London, 1998.

Stahlberger, A., and M. Riedmiller. "Fast network pruning and feature extraction using the unit-OBS algorithm." In M. Mozer, M. Jordan, and T. Petsche, eds., *Neural Information Processing Systems*, pp. 655–661. MIT-Press, 1997.

Starrenburg, J. G., W. T. C. van Luenen, W. Oelen, and J. van Amerongen. "Learning feedforward controller for a mobile robot vehicle." *Control Engineering Practice*, vol. 14, no. 9, pp. 1221–1230, 1996.

Stewart, G. W. *Matrix Algorithms*, vol. 1 : Basic Decompositions. SIAM, Philidelphia, 1998.

Stewart, G. W. *Matrix Algorithms*, vol. II : Eigensystems. SIAM, Philadelphia, 2001.

Stitson, M. O., A. Gammerman, V. N. Vapnik, V. Vovk, C. Watkins, and J. Weston. "Support vector regression with ANOVA decomposition kernels." Tech. Rep. CSD-TR-97-22, Royal Holloway, University of London, 1997.

Suykens, J. A. K., P. van Dooren, B. de Moor, and J. Vandewalle. "Least squares support vector machine classifiers: A large scale algorithm." In *European Conference on Circuit Theory and Design, ECCTD'99*, pp. 839–842. 1999.

Suykens, J. A. K., and J. Vandewalle. "Recurrent least squares support vector machines." *IEEE Transactions on Circuits and Systems - I:Fundamental theory and applications*, vol. 47, no. 7, pp. 1109–1114, 2000.

Tikhonov, A. N., and V. Y. Arsenin. *Solutions of Ill-Posed Problems*. W. H. Winston, Washington D. C., 1977.

Tomizuka, M. "Zero phase error tracking algorithm for digital control." *Journal of Dynamic Systems, Measurement, and Control*, vol. 109, pp. 65–68, 1987.

Unser, M., A. Aldroubi, and M. Eden. "Fast B-spline transform for continuous image representation and interpolation." *IEEE Transactions on pattern analysis and machine intelligence*, vol. 13, no. 3, pp. 277–285, 1991.

Uysal, I., and H. A. Güvenir. "An overview of regression techniques for knowledge discovery." *The Knowledge Engineering Review*, vol. 14, no. 4, pp. 319–340, 1999.

Valkenburg, G. *On Training Strategies for Parsimonious Learning Feed-Forward Controllers*. nr. 001r2001, University of Twente, 2001.

Van Amerongen, J. "Mechatronic design." *Mechatronics*, vol. 13, no. 10, pp. 1045–1066, 2003.

Van de Laar, P., and T. Heskes. "Pruning using parameter and neuronal metrics." *Neural Computation*, vol. 11, pp. 977–993, 1999.

Van Dijk, J., R. Tinsel, and E. Schrijver. "Learning control of direct drive systems with cogging force disturbances." In B. Samali, ed., *Fifth Motion and Vibration Conference, MOVIC2000*, pp. 381–386. Sydney, Australia, 2000.

Van Wermeskerken, B. *Position and Velocity Estimation in Linear Motors*. Master's thesis, University of Twente, 2001. Nr. 014R2001.

Vapnik, V. N. "An overview of statistical learning theory." *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999.

Vapnik, V. N. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 2nd edn., 2000.

Velthuis, W. J. R. *Learning Feed-Forward Control, Theory, Design and Applications*. Ph.D. thesis, University of Twente, Enschede, 2000.

Velthuis, W. J. R., T. J. A. de Vries, K. H. J. Vrielink, G. J. Wierda, and A. Borghuis. "Learning control of a flight simulator stick." *Journal A*, vol. 39, no. 3, pp. 29–34, 1998.

Verwoerd, M. H. A. *Convergence Analysis of Learning Feed-Forward Control*. Master's thesis, University of Twente, 2000. Nr. 026R2000.

Verwoerd, M. H. A., T. J. A. de Vries, and G. Meinsma. "On the use of noncausal LTI operators in iterative learning control." In *41st IEEE Conference on Decision and Control*, pp. 3362–3366. Las Vegas, USA, 2002.

Vijayakumar, S., and S. Schaal. "Locally weighted projection regression : An O(n) algorithm for incremental real time learning in high dimensional spaces." In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pp. 288–293. Stanford, California, 2000.

White, H. "Learning in artificial neural networks: A statistical perspective." *Neural computation*, vol. 1, pp. 425–464, 1989.

White, H. *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell, Cambridge, USA, 1992.

Zurada, J. M. *Introduction to Artificial Neural Systems*. PWS Publishing Company, 1992.