University of Twente

EEMCS / Electrical Engineering
Control Engineering

Variable batch sizes for Learning Feed Forward Control using Key Sample Machines

Job Kamphuis

M.Sc. Thesis

Supervisors
prof.dr.ir. J. van Amerongen
dr.ir. T.J.A. de Vries
dr. ir. B.J. de Kruif

June 2005

Report nr. 024CE2005
Control Engineering
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

Function approximators are able to “learn” a function when presented with a set of training samples. The samples in this training set are generally corrupted by measurement noise. The function approximator will “filter out” this noise and is able to predict the function output value for a new input. Function approximators are used in feed forward controllers. Given a certain input e.g. the reference signal, it is possible to generate a control signal, which makes the physical system behave as dictated by the reference signal.

Function approximators as Least Squares Approximator or Support Vector Machine have been around for some time. During his PhD., dr. ir. B.J. de Kruif developed a new function approximator, which was capable of approximating certain functions that could not be approximated by other algorithms. Furthermore, this approximator, called Key Sample Machine (KSM) approximates the function in an efficient way, saving a lot of computer resources (De Kruif, 2004).

In this project the principals of the two existing variants of KSM are described in detail. The offline variant approximates the function in a non-recursive way. All samples in the training set are presented to the offline KSM at once. When the calculations are done, the offline KSM comes with a final approximation. The online variant approximates the function recursively and handles one training sample at a time. Both KSM algorithms were rebuilt in Matlab and each implementation step has been described in detail in this thesis.

During this project a new KSM algorithm has been developed. This key sample machine is able to handle samples in variable block sizes. It is therefore called “Blocked Key Sample Machine” (BKSM). The target function is approximated recursively, just as the online algorithm, but it can however handle more samples at a time. The handling of multiple samples is very similar to that of the offline key sample machine. The advantage of the BKSM compared to the online KSM is that the approximation is done quicker. Furthermore, the algorithm is contrary to the offline KSM, able to update the approximation continuously. This can be an asset or even a necessity for systems where the target function is not time invariant.

The BKSM algorithm has been tested in the simulation environment of 20-sim (Controllab Products b.v.). The plant is a model of a physical system, which is closely related to a printing device. A motor drives a belt on which a load mass is mounted. The position of the load has to be controlled. First the system is controlled by a PD-controller, which is later on extended with a BKSM feed forward controller. The maximum position error of the PD-controlled system is 2.54 10^{-3} [m]. When the controller is extended with the BKSM, the maximum position error of the system is 1.40 10^{-4} [m]. This means that the error decrease due to the BKSM is about 95 %.

Besides testing the BKSM in simulations, it was tested on a real physical system. This system is called “Mechatronic Demonstrator” (Dirne, 2005). As in the simulations, a PD-controller is used to control the system. This resulted in a maximum position error of 3.10 10^{-3} [m] and a maximum static position error of 3.00 10^{-4} [m]. After the controller was extended with the BKSM, the maximum position error decreased to a value of 4.30 10^{-4} [m]. This means that the maximum position error reduction is 85 %. The maximum static error was reduced by half to a value of 1.50 10^{-4} [m].

From the simulations and the testing on the real physical system, it can be concluded that BKSM is a promising function approximator. However, more research has to be done in testing and benchmarking the BKSM algorithm, in order to assess its contribution to the field of control engineering.

Samenvatting

Functieapproximators zijn in staat een functie te “leren” als er een traingset met samples wordt aangeboden. De samples in deze trainingset zijn over het algemeen besmet met meetruis. De functieapproximator “filtert” deze ruis en is in staat om de functiewaarde te voorspellen voor een gegeven inputwaarde. Functi approximators worden gebruikt in feed forward controllers. Met behulp van een input b.v. het referentiesignaal, is het mogelijk een stuursignaal te generen, waardoor het fysische systeem zich zal gedragen als door het referentiesignaal gedicteerd wordt.

Functieapproximators zoals Least Squares Approximators of Support Vector Machine worden al geruime tijd gebruikt. Tijdens zijn PhD., heeft dr. ir. B.J. de Kruif een nieuwe functieapproximator ontwikkeld, die in staat is om bepaalde functies te benaderen die niet te approximeren waren met behulp van andere algoritmen. Daarnaast approximeert deze functieapproximator, genaamd Key Sample Machine (KSM), functies in een efficiënte manier, waardoor er bespaard wordt op het gebruik van computer bronnen (De Kruif, 2004).

In dit project worden the principes achter the twee bestaande varianten van KSM in detail beschreven. The offline variant approximeert the functie in een niet-recursieve manier. Alle samples in de training set worden in één keer aangeboden aan de offline KSM. Als de berekeningen zijn gedaan, komt de offline KSM met een definitieve approximatie. De online variant approximeert de functie recursief en verwerkt de samples één voor één af. Beide KSM algoritmen zijn opnieuw geprogrammeerd in Matlab en elke stap in de implementatie is in detail beschreven in deze thesis.

Tijdens dit project een nieuw KSM algoritme is ontwikkeld. Deze key sample machine is in staat om samples in variabele blokgrootten te verwerken. Het wordt daarom “Blocked Key Sample Machine” (BKSM) genoemd. De doelfunctie wordt recursief geapproximeerd, zoals in het online algoritme, maar het kan echter meerdere samples tegelijkertijd verwerken. Het verwerken van meerdere samples lijkt erg op dat van de offline key sample machine. Het voordeel van de BKSM vergeleken met het online algoritme is dat de approximatie sneller wordt uitgevoerd. Daarbij is het algoritme in tegenstelling tot de offline KSM, in staat om de approximatie continu te updaten. Dit kan een nuttige eigenschap of zelfs een noodzaak zijn voor systemen waarvan de doelfunctie tijd variant is.

Het BKSM algoritme is getest in de simulatie omgeving van 20-sim (Controllab Products b.v.). Hiervoor wordt een model van een fysisch systeem gebruikt, dat sterk gerelateerd is aan een print machine. Een motor drijft een band aan waarop een massa is gemonteerd. De positie van de massa moet worden gestuurd. Eerst wordt het systeem aangestuurd door een PD-controller, die later wordt uitgebreid met een BKSM feed forward controller. De maximum positie fout van het door de PD-controller gestuurde systeem is 2,54 10^{-3} [m]. Als de controller wordt uitgebreid met de BKSM, wordt de maximale positie fout van het systeem 1,40 10^{-4} [m]. Dit betekent dat de fout verkleining ten gevolge van de BKSM ongeveer 95 % is.

Naast het testen van de BKSM in simulaties, is het getest op een fysisch systeem. Dit systeem wordt “Mechatronic Demonstrator” genoemd (Dirne, 2005). Zoals tijdens de simulaties, wordt eerst een PD-controller gebruikt om het systeem aan te sturen. Dit resulteert in een maximum positie fout van 3,10 10^{-3} [m] en een maximum statische positie fout van 3,00 10^{-4} [m]. Nadat de controller was uitgebreid met de BKSM, bedroeg de maximum positie fout 4,30 10^{-4} [m]. Dit betekent dat de maximum positie fout is gereduceerd met 85 %. De maximum statische positie fout is gereduceerd tot de helft met een waarde van 1.50 10^{-4} [m].

Uit de simulaties en het experimenten op het fysische systeem, kan worden geconcludeerd dat BKSM een veel belovende functie approximator is. Er zal echter meer onderzoek moeten worden gedaan in het testen en benchmarken van het BKSM algoritme om de contributie aan het veld van control engineering te kunnen beoordelen.

Preface

For years I have wished for the moment that my study Electrical Engineering, would come to an end. Now the moment is near, I feel ambivalent and tend to get emotional about it sometimes. There is so much I have learned during my time as a student. Of course I learned a lot about electrical engineering but I even learned a great deal more about myself. The first years have been pretty difficult and I have had several moments, I decided to stop studying electrical engineering. Therefore, the first people I would like to thank are my brother and especially my parents for their ever-lasting support. Second of all I would like to thank my best friend Hans Dirne. Although I would probably have finished this study three years earlier without him, his companionship was essential for having a good time while studying the tough and sometimes tedious subjects. Furthermore I would like to thank two housemates Victor van Eekelen and Joep ter Avest, for their contribution in making our home a great place to live and study.

I am grateful for the supervision of prof. dr. ir. J. van Amerongen and dr. ir. T.J.A. de Vries during this graduation project. They have been given me enough freedom to explore and demonstrate my capabilities as an engineer and also supported me well in the moments I needed some steering. I also would like to thank my third supervisor dr. ir. B.J. de Kruif for the hours he spent with me explaining the abstract mathematical procedures. He is also the main designer of the newly developed blocked key sample machine. I believe that without his help the scope of this project would have been limited severely.

Job Kamphuis
Enschede, June 2005

Table of Contents

1 Introduction

Control engineering is about changing systems dynamics in such a way that the controlled system will behave in a desired way (Van Amerongen, 2001). If the system is well known a precise model can be deduced. With this model it is possible to generate a control signal, which makes the system behave exactly as is dictated by the reference signal ("feed forward control"). However in practice it is impossible to have a model that fully represents the real system. There always will be some elements that are neglected in the model and which result in small differences in states predicted by the model en the real system states. Other differences between model predictions and the system states are caused by external disturbances. This error between reference signal and system states cannot be circumvented. However it can be reduced to a certain level by feedback control, in which the desired and real system states are compared and the error signal is fed back and used to "steer" the system. Using this signal for steering purposes implies that the controller is by definition unable to completely eliminate the error to zero during transients. To achieve further error reduction, the model in the feed forward controller is not made time-invariant but certain parameters in this model can be adjusted while the system is controlled ("learning feed forward control"). Assuming that the process states correspond within certain boundaries, to input references $r, \dot{r}$ and $\ddot{r}$, the part of the error depending on the process states will be reduced by the "u_{ff} signal" depicted in Figure 1.1 (De Kruif, 2004).

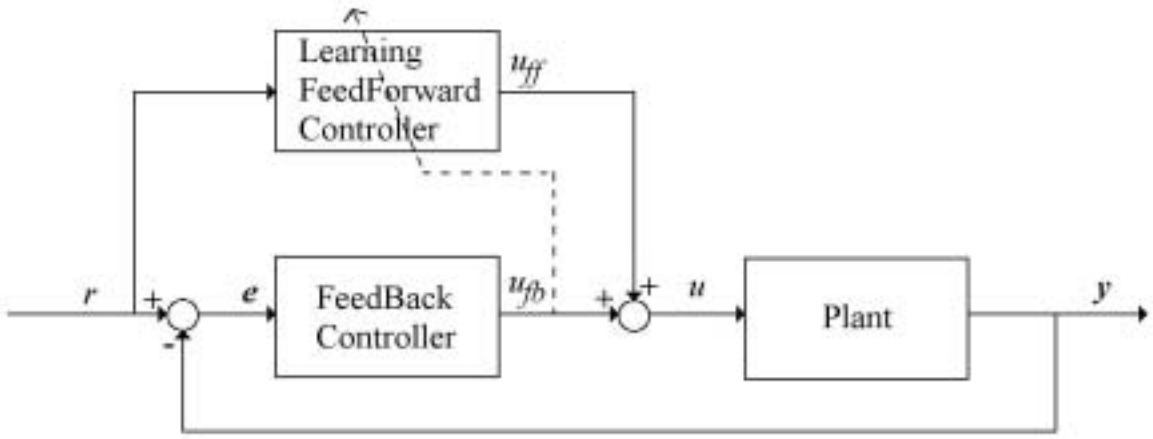

Figure 1.1. Process controlled by feed forward and feedback control

When for example, the position of an electric motor has to be controlled; the feedback controller will control this position with a certain error. A part of this error can be dependent of the system states. For an electric motor the so-called "cogging force" is a property that illustrates this well. The coils have preference positions in relation to magnets and thereby create a cogging force. In Figure 1.2 it is shown how this force could relate to the position of the motor.

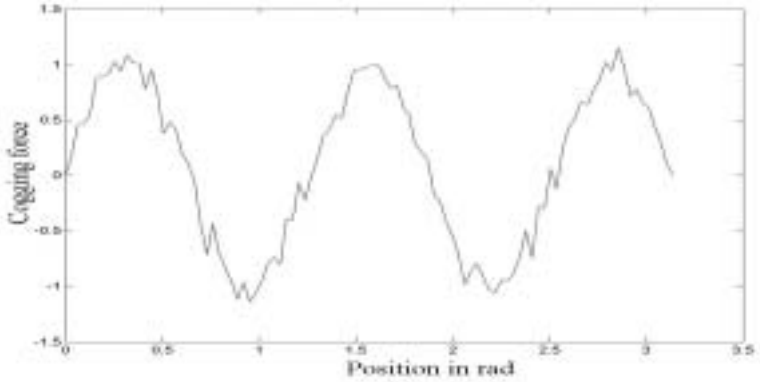

Figure 1.2. Cogging force versus motor position

This force causes a state-dependent error in the position control of the system. However, if the cogging force is known for each position of the motor, the feed forward controller could compensate for it. In general, the feed forward controller can compensate for any state dependent disturbance. In order to do so, it is necessary to approximate the function that relates the concerning state to the accompanying disturbance. Before any approximation can be made, measurements relating states and disturbances have to be taken. These measurements are used as input for a function approximator, which will interpolate between these measurements and so creates a function.

Recently a new algorithm for function approximation has been developed, called “key sample machine” (KSM). This algorithm has a number of advantages compared to previous algorithms like “ordinary least squares”, “dual least squares” or “support vector machines”. KSM´s excel especially in using less computing resources while being more or about as accurate as the algorithms mentioned before. The KSM algorithm has two variants; one approximates the function recursively (“online”), while the “offline” KSM needs all the data in advance.

The main goal of this project is to develop a KSM that forms the midst between the “offline” and “online” algorithms. The purpose of this combined algorithm is that it approximates the target function with some in advance retrieved training samples. Using newly retrieved samples, which can be consumed by the algorithm in variable batch sizes, will increase the accuracy of this preliminary approximation. The project goal can be divided in several sub goals:

- Literature study of existing function approximation algorithms like “least squares”, “support vector machine” and “key sample machine”.
- Development of a new key sample machine algorithm in Matlab
- Programming the new key sample machine algorithm in C or C++
- Testing the new key sample machine by means of simulation
- Implementing and testing of the new key sample machine on a physical system

In chapter 2 two Least Squares function approximation algorithms, which are important for understanding the functioning of KSM, will be treated. The “offline” and “online” KSM algorithms will be explained in detail in chapter 3. In chapter 4 the newly developed Blocked Key Sample Machine (BKSM) is described. The testing of this BKSM is done by simulations and on a real physical system in chapter 5. In chapter 6 the conclusions of recommendations of this project can be found.

2 Least Squares function approximators

In this section two function approximators, which are based on least squares minimization criteria, are treated. The reason for explaining them is because they are commonly known and used. Furthermore the drawbacks of these approximators can be shown and so the reason why key sample machines were developed becomes clearer.

2.1 Ordinary Least Squares function approximator

For LFFC, a function approximator has to approximate functions that relate a disturbance (the target *y*) to certain system-states (the inputs, **x** of the function approximator). The Ordinary Least Squares (OLS) (De Kruif, 2004) method tries to approximate the target by means of a linear relation between certain kernel or indicator functions $f_i(x), i = 1...n$ and the output $\hat{y}$.

$$\hat{y}_{ls}(x) = b_1 f_1(x) + b_2 f_2(x) + ... + b_n f_n(x) \tag{2.1}$$

The set of functions that can be approximated is determined by selecting a set of *n* kernel or indicator functions and an allowable range for parameter vector **b**. Given this set of kernel functions, the OLS algorithm tries to find the value for parameter vector **b** that gives the best approximation measured as a least squares criterion. The structure of (2.1) shows that the approximation problem itself is linear in the parameters.. However, by choosing correct kernel functions it is possible to approximate non-linear functions. The kernel functions implement a mapping from the input space to the so-called feature space. When the set of kernel functions is chosen properly the ´target´ function, which is strongly non-linear in input space, will be (much more) linear in feature space. Therefore it is possible to approximate the non-linear function in a linear way.

To implement the OLS algorithm it is necessary to choose the set of kernel functions. In the example given below the kernel functions are Gaussian functions with variance σ^2 and equidistant centers $c_{i,}$ which are homogeneously spread over the input space:

$$f_i(x) = \exp(-\frac{(x - c_i)^2}{2\sigma^2}) \qquad (i=1..n) \tag{2.2}$$

with

$$c_i = \frac{i}{n-1} - \frac{1}{n-1} \tag{2.3}$$

As an example, we choose example σ = 0.11 and *n* = 10, which means that there will be 10 Gaussian indicator function distributed over input space. Function (2.4) is chosen to generate a training set, which contains 100 samples and is shown in Figure 2.1.

$$y=\text{sinc}(x)*\cos(10*x)+\varepsilon \qquad x \in [0,1] \tag{2.4}$$

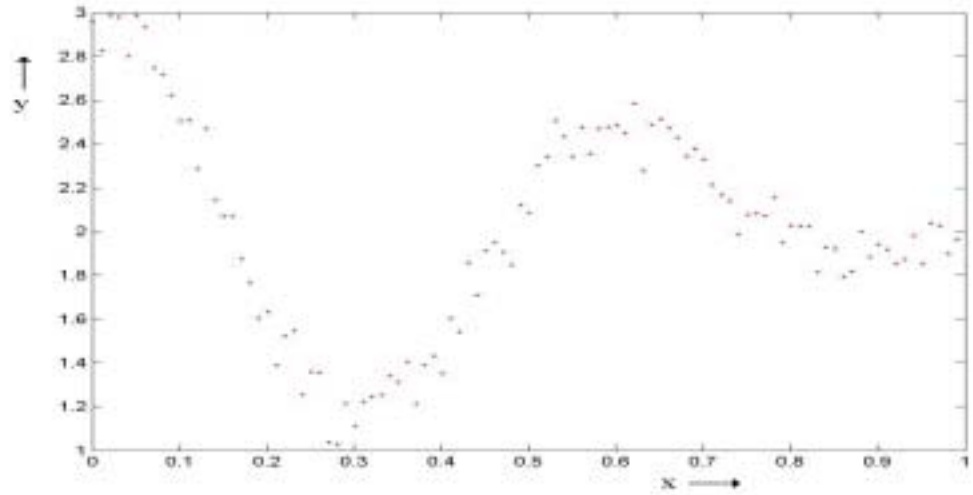

Figure 2.1. Example dataset with 100 data points

With the input data and the kernel function (2.2) indicator matrix **X** is formed; it will have a size of [*N* x *n*] where *N* is the number of data points and *n* is the number indicator functions (in this example *N*=100 and *n* = 10).

$$\mathbf{X} = \begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_N) & f_2(x_N) & \cdots & f_n(x_N) \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2.5}$$

The first elements of this matrix **X** calculated for the example training set are shown below.

$$\mathbf{X} = \begin{bmatrix} 1.00 & 0.61 & 0.14 & 0.01 & 0.00 & \cdots \\ 1.00 & 0.66 & 0.16 & 0.01 & 0.00 & \cdots \\ 0.98 & 0.71 & 0.19 & 0.02 & 0.00 & \cdots \\ 0.96 & 0.76 & 0.22 & 0.02 & 0.00 & \cdots \\ 0.93 & 0.82 & 0.26 & 0.03 & 0.00 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.6}$$

Vectors **x** and **y** are input data and the latter is assumed to be corrupted by normally distributed noise, **ε**. By using the matrices introduced as above the following relation holds:

$$\mathbf{y} = \mathbf{Xb} + \varepsilon + \phi = \mathbf{Xb} + \mathbf{e} \tag{2.7}$$

In this formula **ε** is the normally distributed noise on the samples and **ϕ** is the approximation error, which depends on the choice of the indicator function. The total approximation error **e**, is the sum of **ε** and **ϕ.** The minimization of the summed squared approximation error is given as:

$$\min_b \left(\frac{1}{2} \| \mathbf{Xb} - \mathbf{y} \|_2^2 + \frac{1}{2} \lambda \| \mathbf{b} \|_2^2\right) \tag{2.8}$$

In which the 2-norm (Stewart, 1998) is defined as:

$$\|x\|_2 = \sqrt{\sum_i |x_i|^2} \tag{2.9}$$

The term $\frac{1}{2}\| \mathbf{b} \|_2^2$ is the so-called regularisation term that minimises the parameters and λ is a positively valued regularisation parameter. Taking the derivative to **b** of formula (2.8) and equating this derivative to zero yields:

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{b} = \mathbf{X}^T\mathbf{y} \tag{2.10}$$

or

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.11}$$

In this equation **I** is the identity matrix. By solving this equation the vector **b** is calculated and will have the size of [*n* x 1]. The values of **b** for the example are given below.

$$\mathbf{b} = \begin{bmatrix} 2.35 & 0.92 & 0.31 & 0.33 & 0.63 & 1.28 & 0.99 & 0.81 & 0.41 & 1.47 \end{bmatrix}^T \tag{2.12}$$

With parameter vector **b** known, it is possible to give an estimate for a new sample $\boldsymbol{x_{new}}$.

$$\hat{y}(\mathbf{x}_{\mathbf{n}ew}) = \sum_{i=1}^{n} b_i f_i(\mathbf{x}_{\mathbf{n}ew}) = \mathbf{f}(\mathbf{x}_{new})\mathbf{b} \tag{2.13}$$

The parameter vector **b** contains the weights of the Gaussian indicator functions, which have been drawn as vertical bars in Figure 2.2. By summing the weighted evaluations of the indicator functions the approximation is constructed. The samples in the training set are depicted by the (red) dots. As one can see for this particular situation a good approximation is obtained.

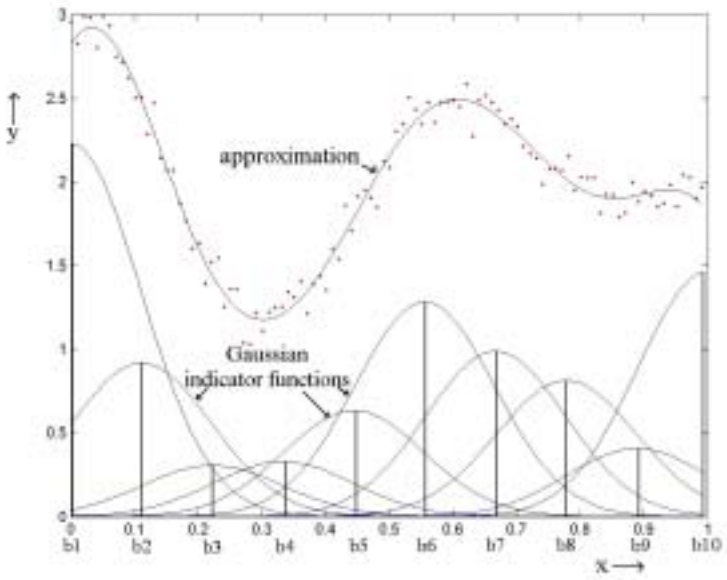

Figure 2.2. Data set, OLS approximation accompanying Gaussian splines

2.2 Dual Least Squares function approximator

The Ordinary Leas Squares algorithm divides the input space into regions, as a result of the choice of the kernel functions. In the example of section 2.1 the input space ($x \in [0,1]$), was divided into ten different regions by the Gaussian indicator functions. The disadvantage with this approach is that every part of the input space is mapped into feature space even if there are not any data points present in that part. Hence, although no data may be given for a part of the input space, one does have to take that part of the input space into consideration. Putting the minimization problem of (2.8) in a dual form helps to circumvent this non-data-driven mapping from input to feature space. The minimization problem can be written as (De Kruif, 2004):

$$\min(\frac{1}{2}\mathbf{e}^T\mathbf{e} + \frac{1}{2}\lambda\mathbf{b}^T\mathbf{b}) \tag{2.14}$$

with

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b} \tag{2.15}$$

in which **e** is a column-vector containing the approximation error for all the samples. When e is minimized, ideally it consists of only the noise, which contaminated the training set. This means that and in this case **e=**ε from (2.7). From (2.14) a Langrangian can be constructed:

$$L = \frac{1}{2}\mathbf{e}^T\mathbf{e} + \frac{1}{2}\lambda\mathbf{b}^T\mathbf{b} + \mathbf{a}^T(\mathbf{y} - \mathbf{X}\mathbf{b} - \mathbf{e}) \tag{2.16}$$

The vector **a** consists of the Lagrangian multipliers and has the same dimension, *N*, as the number of samples. The minimisation problem (2.14) can be solved by equating the derivatives of the Langrangian with respect to **b**, **e** and **a** to zero. The solution of (2.16)
gives the expression:

$$(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})\mathbf{a} = \mathbf{y} \tag{2.17}$$

In section 2.1, the solution was given as:

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\mathbf{b} = \mathbf{X}^T\mathbf{y} \tag{2.18}$$

These solutions are identical if the following holds.

$$\mathbf{b} = \mathbf{X}^T\mathbf{a} \tag{2.19}$$

The estimation of output for the new input values can be calculated as follows:

$$\hat{y}(\mathbf{x}_{new}) = \mathbf{f}(\mathbf{x}_{new})\mathbf{b} = \mathbf{f}(\mathbf{x}_{new})\mathbf{X}^T\mathbf{a} \tag{2.20}$$

in summation form, this can be written as:

$$\hat{y}(\mathbf{x}_{new}) = \sum_{i=1}^{n} b_i f_i(x_{new}) = \sum_{i=1}^{N} \langle \mathbf{f}(\mathbf{x}_{new}), \mathbf{f}(\mathbf{x}_i) \rangle a_i \tag{2.21}$$

Where $\langle \mathbf{f}(\mathbf{x}_{new}), \mathbf{f}(\mathbf{x}_i) \rangle$ is the inner product of vector $\mathbf{f}(\mathbf{x}_{new})$ and $\mathbf{f}(\mathbf{x}_i)$ and in which vector $\mathbf{f}(\mathbf{x}_i)$ contains the values of all the kernel functions with input the *i-th* sample from vector **x**.

The point to observe here is that calculation of the inner product does not require explicit calculation of $\mathbf{f}(\mathbf{x}_{new})$. Hence the prediction with the use of **a** is not depending on the number of kernel functions but only on the number of samples *N*. Therefore, it is possible to use a large or even an infinite dimensional feature space for the approximation. When training with data sets that are relatively small and not homogenously spread over input space, the DLS algorithm will probably be more efficient than the OLS algorithm. However, the data contained by vector **a** in the DLS algorithm is abstract compared to that of vector **b** in the OLS algorithm and therefore it is more difficult to check whether the implemented DLS algorithm works properly.

However dual and ordinary least squares function approximators are commonly used they are not able to approximate certain functions. This training set of these functions is corrupted by measurement noise and the function itself has large differences in its derivative values. In Figure 2.3 it the least squares algorithm is unable to approximate the function in areas, which it has, large absolute values for its derivative. Furthermore in the area where the derivative is small the algorithm starts to fit the measurement noise.

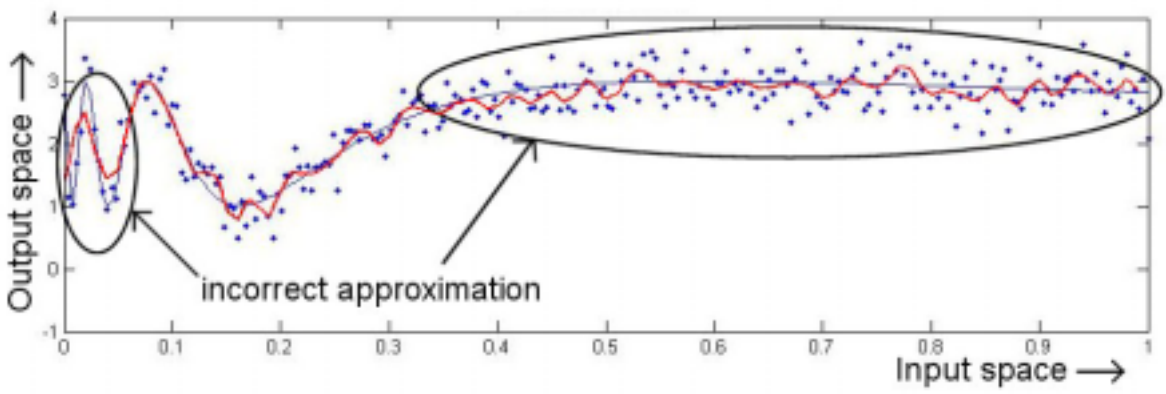

Figure 2.3. Least squares approximation

The trainings set consists of samples, which in the figure are depicted as dots. The approximation is the thicker (red) line. In the input area between 0 and 0.1 the function has large absolute values for its derivative and here the least squares approximator is unable to “follow” the function. On large values of the input space the derivative values are small and here the approximator fits the measurement noise into its approximation. This is one of the reasons for developing Key Sample Machine approximators.

3 Existing Key Sample Machine algorithms

Key Sample Machines were developed to approximate functions in a more sophisticated way compared to ´Ordinary Least Squares´, ´Dual Least Squares´ or even ´Support Vector Machines´. The goal was to develop an algorithm that (De Kruif, 2004):

- *Uses the summed squared approximation error as cost function*

- *Uses a sample-based approach in which the prediction parameters are trained by using the inner product between the samples*

- *Uses some of the samples for the prediction, but use all the samples for the training (data compression)*

- *Uses a subset selection scheme that fits the problem at hand to select the samples that are to be used for a proper prediction.*

The KSM algorithm combines advantages of several function approximation algorithms and is therefore more accurate, less time consuming or less demanding on computer-resources.

3.1 Offline Key Sample Machine

The offline Key Sample Machine (KSM) makes an approximation of the target function based on all samples in the training set (De Kruif, 2004). The first step is to treat all samples as potential key samples. That means that for every sample i, there is a corresponding kernel function $\mathbf{f}_i$. The kernel functions span the so-called feature space into which all samples are projected. So if the training set consists of N samples it results in a matrix with dimension [N x N] in which the projected sample information is contained. This matrix is called the potential indicator matrix $\mathbf{X}_{\text{potential}}$:

$$\mathbf{X}_{potential} = \begin{bmatrix} f_1(\mathbf{x}_1) & f_2(\mathbf{x}_1) & \cdots & f_N(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_N) & f_2(\mathbf{x}_N) & \cdots & f_N(\mathbf{x}_N) \end{bmatrix} \tag{3.1}$$

In this matrix each column represents a potential key sample. These potential key samples have to be assessed, so a selection can be made which potential key samples will become “real” key samples. The latter will be used in the approximation of the function. The “real” key samples n, will, the construct indicator matrix $\mathbf{X}$ [N x n]:

$$\mathbf{X} = \begin{bmatrix} f_1(\mathbf{x}_1) & f_2(\mathbf{x}_1) & \cdots & f_n(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_N) & f_2(\mathbf{x}_N) & \cdots & f_n(\mathbf{x}_N) \end{bmatrix} \tag{3.2}$$

The columns of this matrix contain the information about the key samples after projection into the feature space. Each column corresponds to a key sample that in its turn determines a unique kernel function. For instance if the selected kernel function is a 1st order B-spline then the position of its centre is determined by the key sample. This is illustrated in Figure 3.1.

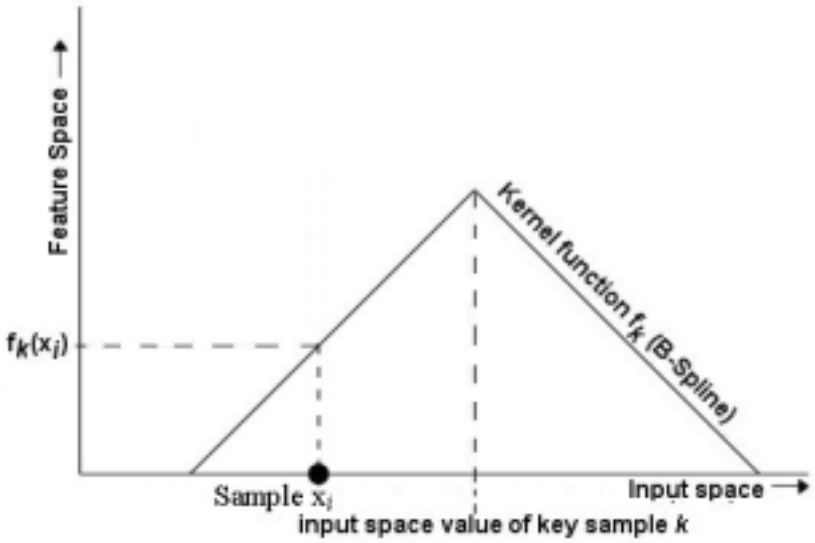

Figure 3.1. Projection of a sample into feature space by a B-spline kernel function

To approximate the target function, the kernel functions, which are characterized by the key samples, are given a weight. In Figure 3.2, four key samples are used for the approximation. This means that four kernel functions (in this particular case B-splines) are defined and each has a certain weight (w1…w4). By summing the kernel functions the output values of the target function are predicted.

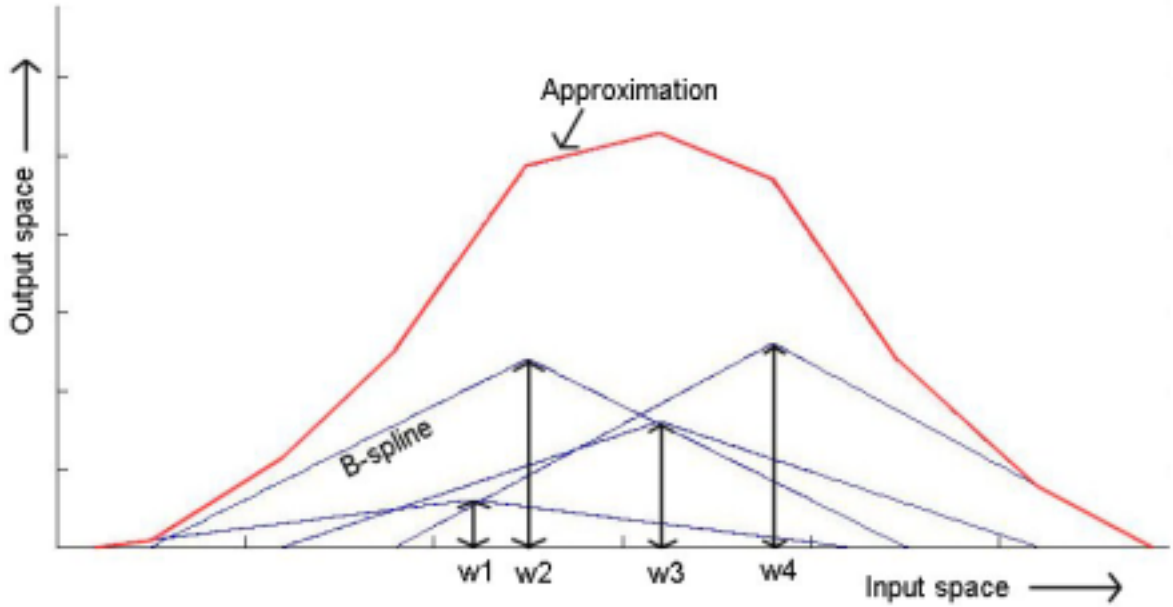

Figure 3.2. Function approximated with four B-spline kernel functions

Each potential key sample has to be assessed whether it has to become a key sample, which is used for the approximation. The error reduction that will result when promoting a potential key sample to an actual key sample is calculated for each potential key sample. When the error reduction is smaller than a certain threshold value, which is related to the estimated noise on the target function, the set of key samples is complete. The weights of the key samples are determined by taking all samples into account.

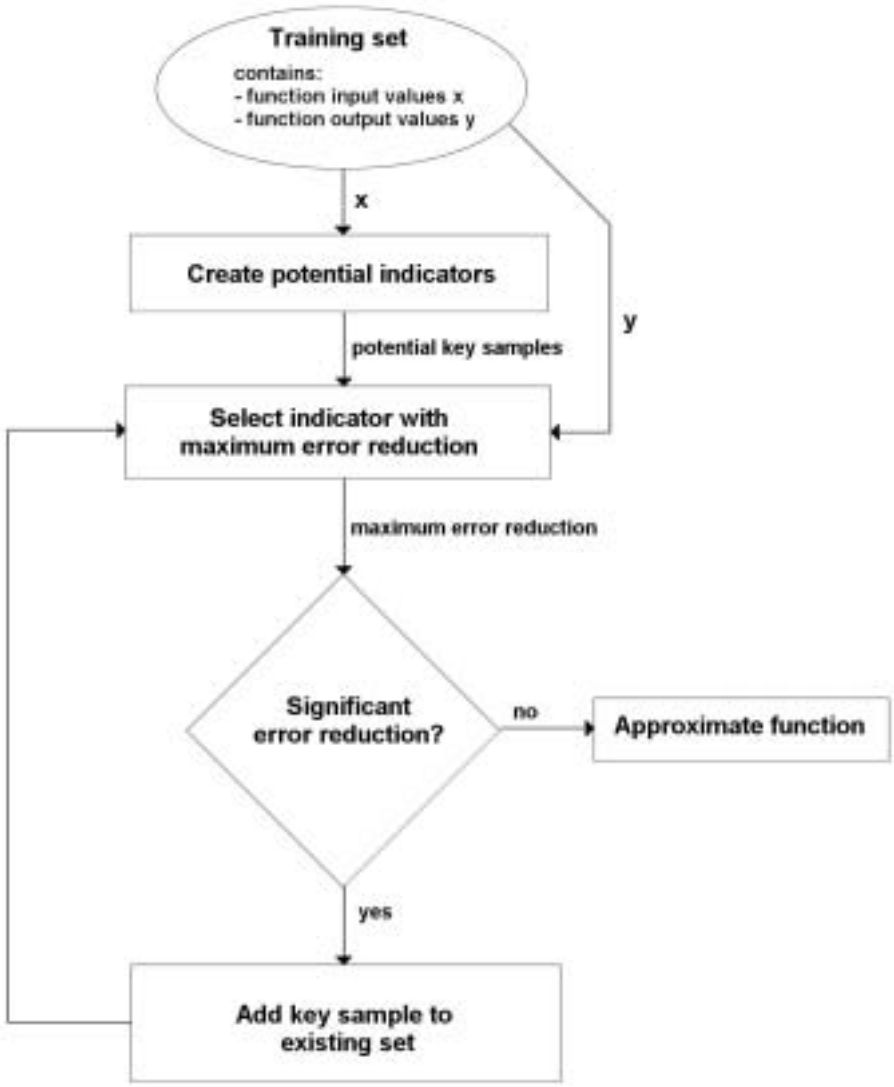

Figure 3.3. Simplified block diagram of the offline KSM algorithm

In Figure 3.3, a simple block diagram of the offline KSM algorithm is depicted. All steps of this algorithm will be treated in more detail below.

3.1.1 Training Set

First a dataset with training samples is needed. This training set contains *N* samples, which are stored in the input and output vector (**x** and **y**). When evaluating the performance of an algorithm, it is important to choose a proper training set. By choosing a training set, which has a large difference in absolute value of its derivative, it is possible to test whether the approximator can handle different fluctuation speeds or not. Formula (3.3) can be used to create a training set with such characteristics.

$$y = \sin(\frac{1}{x+v}) + \varepsilon \tag{3.3}$$

In which ε is noise with zero mean and a Gaussian distribution. This relation is plotted in Figure 3.4. with ε = 0 and v = 0.05

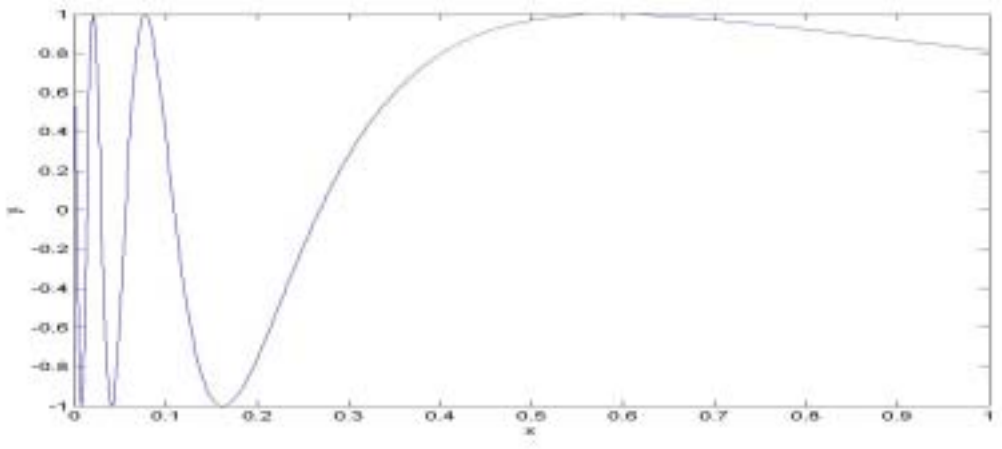

Figure 3.4. Function with large differing derivative values

3.1.2 Create potential indicators

In the process of creating potential indicators two steps can be distinguished. First the samples have to be projected into feature space. Second the projected samples are used for creating potential indicators.

3.1.2.1 Kernel function

The input vector **x** is projected into feature space by selected kernel functions. Any function can be selected as kernel function. However, by selecting kernel functions, the total set of possible functions that can be approximated, is determined. It is important that the function that has to be approximated is an element of the set of possibilities. In other words, the kernel functions have to be chosen in a way such that the target function is linear in feature space. In most cases, the structure of the target function is unknown; therefore kernel functions giving a large set of possibilities are preferred. Infinite splines or B-splines can approximate any function that has a certain maximum value for its derivative. The formula for first order B-splines is given as a convolution below:

$$f^1(x-x_c) = f^0(x-x_c) * f^0(x-x_c) \tag{3.4}$$

with

$$f^0 = \begin{cases} 1 & if -0.5 \le x - x_c < 0.5 \\ 0 & elsewhere \end{cases} \tag{3.5}$$

The first order B-spline function has a triangular shape with its centre at x_c. In the same way as in the example in section 2, altering the height of the splines can approximate the target function. In the OLS algorithm the whole input space is projected into feature space. In the DLS algorithm, that part of the input space that contains data points is projected. In KSM, this dual projection is also applied. Each sample in the training set represents a potential indicator function, which is in this case a B-spline. The centres of these splines correspond to the x-values of the samples in the training set. Therefore, there are no indicator functions in those parts of the input space that do not contain any samples.

The notation of a dual indicator function is given below.

$$\tilde{f}_i(\mathbf{x}) = f(\mathbf{x}, \mathbf{x}_i) \tag{3.6}$$

The dual indicator function $\tilde{f}$ will project all samples **x**, from input space to that part of feature space, which is spanned by sample $\mathbf{x}_i$. In case of B-spline indicator functions, $\mathbf{x}_i$ is the same as x_c in formula (3.4). In Figure 3.5 a data set with four training samples is drawn. Each sample is considered as a potential key sample. Therefore all samples generate a potential B-spline indicator function, indicated with A, B, C and D in this figure. Each sample is projected on each B-spline. For example, the value A1 (on the y-axis) stands for the y-value of the projection of sample 1 on B-spline A.

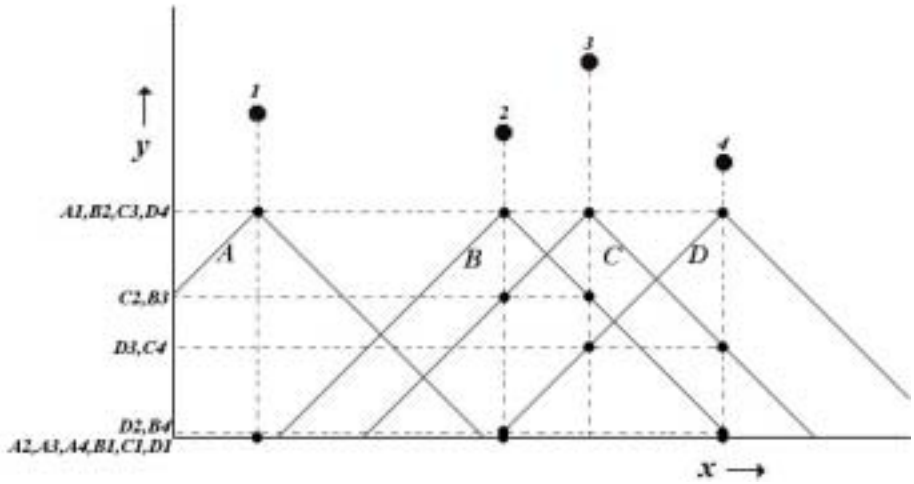

Figure 3.5. Training set with four samples and four potential B-spline indicator functions

3.1.2.2 Creating potential indicator matrix (Xpotential)

The projections from input to feature space are stored in a matrix called $\mathbf{X}_{\text{potential}}$. In this matrix every column represents a key sample, which potentially can be used for the approximation.

$$\mathbf{X}_{potential} = \begin{bmatrix} \tilde{f}_1(\mathbf{x}_1) & \tilde{f}_2(\mathbf{x}_1) & \cdots & \tilde{f}_n(\mathbf{x}_1) \\ \tilde{f}_1(\mathbf{x}_2) & \tilde{f}_2(\mathbf{x}_2) & \cdots & \tilde{f}_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_N) & \tilde{f}_2(\mathbf{x}_N) & \cdots & \tilde{f}_n(\mathbf{x}_{N)} \end{bmatrix} \tag{3.7}$$

3.1.3 Select indicator with maximum error reduction

Before adding one of the potential key samples to the real set of key samples, it is necessary to know which potential key sample will reduce the approximation error the most. This selection procedure consists of two steps, which are treated in detail in this paragraph.

3.1.3.1 Determining the residual

In case of the data set depicted in Figure 3.5 the potential indicator matrix contains the following data.

$$\mathbf{X}_{potential} = \begin{bmatrix} A1 & B1 & C1 & D1 \\ A2 & B2 & C2 & D2 \\ A3 & B3 & C3 & D3 \\ A4 & B4 & C4 & D4 \end{bmatrix} \tag{3.8}$$

Determining which potential key sample has to become a real key sample is done as follows. The residual **e**, is the difference between the function output values of the trainings set (**y**) and the predictions (**Xb**) of the function output values:

$$\mathbf{e} = \mathbf{y} - \mathbf{Xb} \tag{3.9}$$

The matrix **X** is called the indicator matrix and contains the previously selected key samples. Initially it contains only the value one and has the size of [N x 1]. Initially adding key samples to the matrix **X** results in a more accurate approximation and therefore a smaller residual. However, when all samples are included as key samples, noise is being fitted into the approximation. Therefore there is an optimum in the number of samples that are used as key samples. In practice there always is a demand for a certain level of accuracy of the approximation. The optimal solution will be to include the minimal amount of key samples in the matrix **X** that suffices to approximate the target function with the demanded accuracy. This can be done by recursively including a potential key sample that gives the largest reduction of the residual.

3.1.3.2 Calculating the angle between indicator vectors and the residual

The column vectors ($\mathbf{a}_i$), in matrix $\mathbf{X}_{potential}$ represent the potential key samples in feature space. By calculating the angle between these potential key samples and the vector **e**, it is possible to determine which potential key sample will reduce the residual most when added to **X**. This is shown in Figure 3.6.

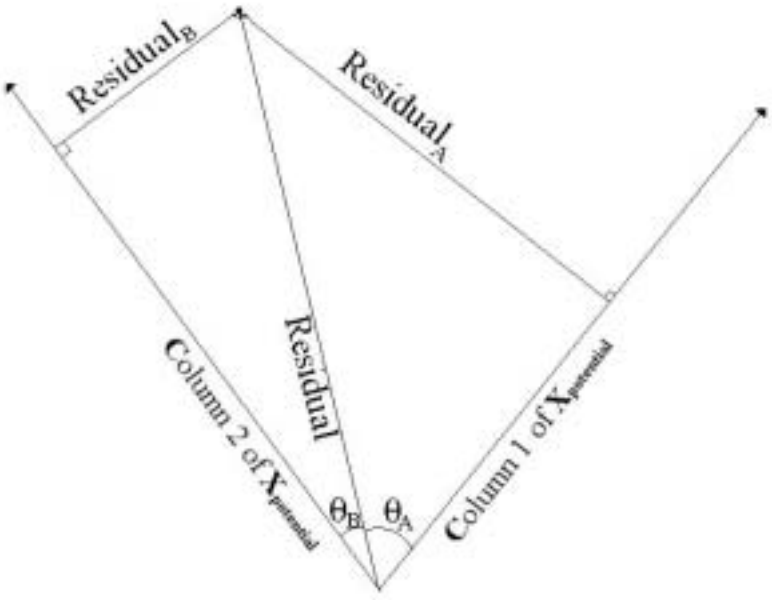

Figure 3.6. Angle between key samples and residual

In this figure two key samples and the current residual are drawn. $Residual_A$ and $Residual_B$ are the remaining residuals after inclusion of respectively key sample 1 or 2. The angle between the key samples and the residual contains information to determine which key sample is will reduce the residual most when added to the indicator matrix. A small angle results in a large decrease of the

residual. Ideally the angle is zero and the residual will be entirely eliminated but this will in general not be the case. The angle between the residual and the key samples can be calculated with formula (3.10).

$$err_i = \cos(\theta)^2 = \frac{(\mathbf{e}^T \mathbf{a}_i)^2}{\mathbf{a}_i^T \mathbf{a}_i (\mathbf{e}^T \mathbf{e})} \tag{3.10}$$

In which err_i is the normalised change in error due to the inclusion of potential indicator *i*. The vector $\mathbf{a}_i$ is the *i-th* column vector in $\mathbf{X}_{potential}$, corresponding to the indicator defined by the *i-th* key sample. The key sample that results in the largest decrease in error should be selected for inclusion.

3.1.4 Add key sample to existing set

For a stable and relative fast inclusion of a key sample it is necessary to apply QR-decomposition onto matrix **X**. First this decomposition will be explained and second the actual inclusion will be treated.

3.1.4.1 QR-decompositon

The selected potential indicator ($\mathbf{a}_{max\ i}$) will be added to indicator matrix **X**. With this ´updated´ matrix the vector **b**, which is used for the approximation of the function, can
be calculated in the same way as in the OLS algorithm.

$$(\mathbf{X}^T \mathbf{X})\mathbf{b} = \mathbf{X}^T \mathbf{y} \tag{3.11}$$

Calculating vector **b** with this formula is demanding on computer resources, because the inverse of ($\mathbf{X}^T\mathbf{X}$) has to be calculated. By applying QR-decomposition on the indicator matrix X, vector **b** can be calculated without explicitly determining this inverse and therefore extensive calculations are evaded. If **X** is of full column rank, QR-decomposition implies that that:

$$\mathbf{X} = \begin{bmatrix} \mathbf{Q}_X & \mathbf{Q}_\perp \end{bmatrix} \begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix} \tag{3.12}$$

In which $\mathbf{Q}_X$ forms an orthogonal base for the column vector space of **X** and $\mathbf{Q}_\perp$ an orthogonal base for the null space of **X**. Matrix **R** is an upper triangular matrix in which the information about the relation between **X** and **Q** is stored. For example, suppose the training set contains three samples from which the first two samples are key samples. In Figure 3.7 this situation is sketched with first order B-splines as kernel function.

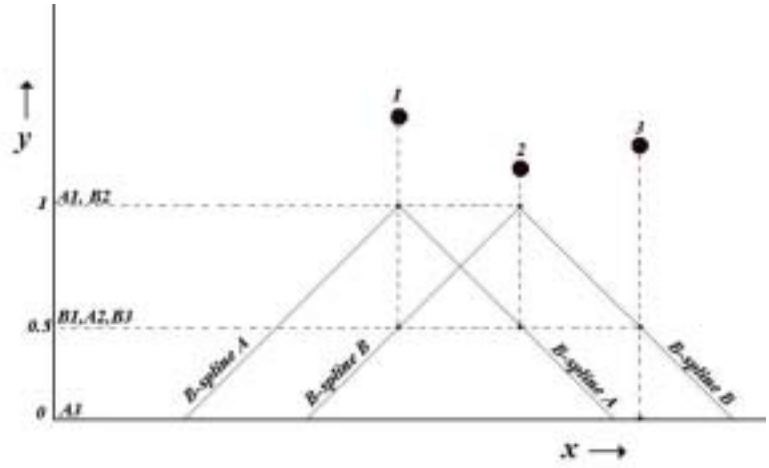

Figure 3.7. Sketch of data set with three samples and two B-splined key samples

The indicator matrix **X** will be in the form:

$$\mathbf{X} = \begin{bmatrix} A1 & B1 \\ A2 & B2 \\ A3 & B3 \end{bmatrix} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \\ 0 & 0.5 \end{bmatrix} \tag{3.13}$$

The feature space, which is spanned by the column vectors of **X** is two dimensional. This 2-dimensional feature space is spanned within 3-dimensional row space in which the axes are respectively $\tilde{f}_i(x_1), \tilde{f}_i(x_2), \tilde{f}_i(x_3)$ with $i \in (1,2)$. The vector space of $\mathbf{Q_X}$ forms an orthonormal base for the feature space and $\mathbf{Q_\perp}$ forms the orthonormal base for the null space of **X.** This is drawn in Figure 3.8. The column vectors of **Q** have been drawn in the correct direction but the lengths are not normalized to avoid an indistinct figure.

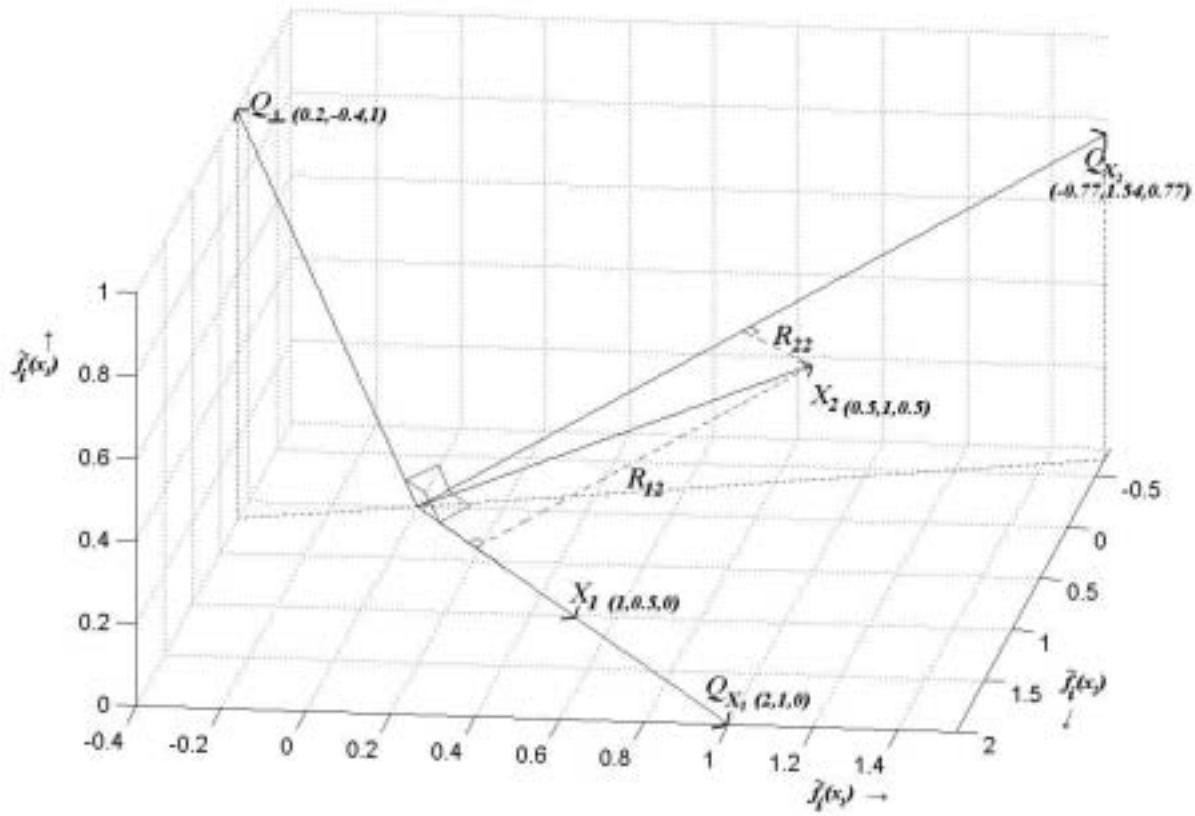

Figure 3.8. Column space of X (expression (3.13)) and column space of accompanying Q

The vector $\mathbf{Q_{X1}}$ has per definition the same orientation as vector $\mathbf{X_1}$. Vector $\mathbf{Q_{X2}}$ is perpendicular to $\mathbf{Q_{X1}}$ but will be in the same plane as $\mathbf{X_1}$ and $\mathbf{X_2}$. This plane represents the feature space and is perpendicular to the null space $\mathbf{Q_\perp}$. The matrix **R** is an upper-triangular matrix, from which element (1,1) is the inner product between $\mathbf{Q_{X1}}$ and $\mathbf{X_1}$. Because all $\mathbf{Q_X}$ column vectors are orthonormal the other elements in the first column of **R** will be zero. The second column of **R** will only have non-zero values on the first two elements. In Figure 3.8 the vector $\mathbf{X_2}$ is projected on $\mathbf{Q_{X1}}$ and $\mathbf{Q_{X2}}$**.** The projection values are stored in respectively $\mathbf{R_{12}}$ and $\mathbf{R_{22}}$. The other elements in this column will be zero because of the fact that the column space of **Q** is orthonormal. Via this reasoning it can be seen that **R** will be an upper-triangular matrix by construction.

For the indicator matrix in (3.13) a possible QR-decomposition could be:

$$\mathbf{Q} = \begin{bmatrix} -0.8944 & 0.3568 & 0.2673 \\ -0.4472 & -0.7171 & -0.5345 \\ 0 & -0.5976 & 0.8018 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} -1.1180 & -0.8944 \\ 0 & -0.8367 \\ 0 & 0 \end{bmatrix} \tag{3.14}$$

The first column vector of **Q** has the same direction as the first column vector of **X**. Element (1,1) of **R,** is the projection of the first column vector of **X** onto the first column vector of **Q**. The second column vector of **Q** is orthonormal to the first column vector of **Q**. Therefore Element (2,1) of **R**, which is the projection of the first column vector of **X** onto the second column vector of **Q**, is zero. The rank of **X** is two for this example, however the two column vectors are defined in a three-dimensional space. So the orthonormal basis of **Q** also spans a three dimensional space. The column vectors of **X** can maximally span a two-dimensional space, so that means that one dimension spanned by the column vectors of **Q** is the so-called null space. This space cannot be "reached" by any combination of the two column vectors of **X**. In Figure 3.8, this null space is spanned by $\mathbf{Q}_\perp$. The residual vector, which is calculated in paragraph 3.1.3.1, is for this example defined in the same three-dimensional space as **X.** When this residual vector has a non-zero element within the null space it is impossible to reduce this part of the error by means of the existing key samples. If this error is larger than the accepted error, a new key sample has to be added. By adding this key sample the rank of **X** will become three and therefore there is no null space. The residual vector can be reduced until it contains only the measurement noise.

In the example above the training set consists of three samples. In practice training sets will consist of about 100-100.000 samples. If the training set consists of N samples of which there are n key samples selected the indicator matrix **X** will be of the size [N x n]. Because of the angle calculation with which key samples are selected, key samples will never be identical and therefore never generate the same column vector. This means that the rank of **X** will be always equal to n. The null space $\mathbf{Q}_\perp$ will be equal to N-n, which is also true for the example. If the indicator matrix has the form:

$$\mathbf{X} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N1} & X_{N2} & \cdots & X_{Nn} \end{bmatrix} \tag{3.15}$$

Q will have the form:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_X & \mathbf{Q}_\perp \end{bmatrix} = \underbrace{\begin{bmatrix} Q_{11} & Q_{12} & \cdots & Q_{1n} & Q_{1(n+1)} & \cdots & Q_{1N} \\ Q_{21} & Q_{22} & \cdots & Q_{2n} & Q_{2(n+1)} & \cdots & Q_{2N} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ Q_{N1} & Q_{N2} & \cdots & Q_{Nn} & Q_{N(n+1)} & \cdots & Q_{NN} \end{bmatrix}}_{Q_{\mathbf{X}} \qquad\qquad Q_\perp} \tag{3.16}$$

and **R** will be:

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ R_{21} & R_{22} & \cdots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ R_{n1} & R_{n2} & \cdots & R_{nn} \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ 0 & R_{22} & \cdots & R_{2n} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & R_{nn} \end{bmatrix} \tag{3.17}$$

X will always be of full column rank, because identical potential key samples will not be selected to be included in **X**. The solution of the least squares problem of minimizing $\| \mathbf{y} - \mathbf{Xb} \|_2^2$ will be uniquely determined by the QR equation

$$\mathbf{Rb} = \mathbf{Q}_X^T \mathbf{y} \equiv \mathbf{z}_\mathbf{X} \tag{3.18}$$

The goal is to calculate **b** without using formula (3.11). From the formula above it can be seen that **b** can be calculated by:

$$\mathbf{b}=\mathbf{z_x}/\mathbf{R} \tag{3.19}$$

Augmenting **X** with **y** and then performing QR-decomposition on this augmented matrix $\mathbf{X_a}$ makes it possible to calculate $\mathbf{z_x}$. The QR-decomposition of $\mathbf{X_a}$ is shown below:

$$\mathbf{X_a}=\mathbf{Q_a}\mathbf{R_a} \tag{3.20}$$

$$\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} = \begin{bmatrix}\mathbf{Q} & \dfrac{\mathbf{r}}{\| \mathbf{r} \|_2}\end{bmatrix}\begin{bmatrix}\mathbf{R} & \mathbf{z}_\mathbf{X} \\ 0 & \| \mathbf{r} \|_2\end{bmatrix} \tag{3.21}$$

The **R** matrix of the QR-decomposition can be found by means of Cholesky decomposition, without the necessity to calculate the corresponding **Q.** When **X** has a full column rank, then the Cholesky decomposition of $\mathbf{X}^T\mathbf{X}$ exists:

$$\mathbf{X}^T\mathbf{X} \;=: \mathbf{R}_{chol}^T \mathbf{R}_{chol} \tag{3.22}$$

Substitution of the QR-decomposition for **X** in $\mathbf{X}^T\mathbf{X}$ yields:

$$\mathbf{X}^T\mathbf{X} = \mathbf{R}^T\mathbf{Q}^T\mathbf{Q}\mathbf{R} = \mathbf{R}^T\mathbf{R} \tag{3.23}$$

From which it follows that the Cholesky decomposition of $\mathbf{X}^T\mathbf{X}$ equals the **R** of the QR-decomposition of **X**, because both are upper-triangular.

$$\mathbf{R} = (\mathbf{X}^T\mathbf{X})_{chol} \tag{3.24}$$

The advantage is that the matrix **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 $\mathbf{X_a}$**.** By means of Cholesky decomposition the $\mathbf{R_a}$ and the corresponding $\mathbf{z_x}$ are found. With these, the parameter vector **b** can be calculated.

3.1.4.2 Inclusion of a key sample in QR-decomposition

When a potential key sample is selected as candidate to become a ´real´ key sample, the indicator matrix **X** has to be extended with one column. As seen in section 3.1.1 the Cholesky decomposition of $(\mathbf{X}_a^T\mathbf{X}_a)$ results in $\mathbf{R_a}$. To circumvent calculating the Cholesky decomposition each time a key sample is added to **X** it is necessary to know in what way $\mathbf{R_a}$ changes. Before the update relation (3.24) holds, so that:

$$\mathbf{X}_a^T\mathbf{X}_a = \mathbf{R}_a^T\mathbf{R}_a \tag{3.25}$$

when written out, the result is:

$$\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}^T\begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix}=\begin{bmatrix}\mathbf{R} & \mathbf{r}\\ 0 & \rho\end{bmatrix}^T\begin{bmatrix}\mathbf{R} & \mathbf{r}\\ 0 & \rho\end{bmatrix} \tag{3.26}$$

$$\begin{bmatrix}\mathbf{X}^T\mathbf{X} & \mathbf{X}^T\mathbf{y}\\ \mathbf{y}^T\mathbf{X} & \mathbf{y}^T\mathbf{y}\end{bmatrix}=\begin{bmatrix}\mathbf{R}^T\mathbf{R} & \mathbf{R}^T\mathbf{r}\\ \mathbf{r}^T\mathbf{R} & \mathbf{r}^T\mathbf{r}+\rho^2\end{bmatrix} \tag{3.27}$$

This gives the following equalities:

$$\mathbf{R^TR=X^TX} \tag{3.28a}$$
$$\mathbf{R^Tr=X^Ty} \tag{3.28b}$$
$$\mathbf{y^Ty=r^Tr}+\rho^2 \tag{3.28c}$$

After the update $\mathbf{X_a}$ will have the form:

$$\overline{\mathbf{X}}_a=\begin{bmatrix}\mathbf{X} & \mathbf{x} & \mathbf{y}\end{bmatrix} \tag{3.29}$$

In which **X** is the existing indicator matrix, **x** is the key sample to be added and **y** is the vector that contains all the output values of the training set. Then $\mathbf{R}_a$ will be:

$$\overline{\mathbf{R}}_a=\begin{bmatrix}\overline{\mathbf{R}} & \mathbf{g} & \overline{\mathbf{r}}\\ 0 & \gamma & \overline{\rho}\\ 0 & 0 & \tau\end{bmatrix} \tag{3.30}$$

After the update the following relation still has to hold:

$$\overline{\mathbf{X}}_a^T\overline{\mathbf{X}}_a=\overline{\mathbf{R}}_a^T\overline{\mathbf{R}}_a \tag{3.31}$$

If (3.29) and (3.30) are filled in (3.31) the result is:

$$\begin{bmatrix}\mathbf{X}^T\mathbf{X} & \mathbf{X}^T\mathbf{x} & \mathbf{X}^T\mathbf{y}\\ \mathbf{x}^T\mathbf{X} & \mathbf{x}^T\mathbf{x} & \mathbf{x}^T\mathbf{y}\\ \mathbf{y}^T\mathbf{X} & \mathbf{y}^T\mathbf{x} & \mathbf{y}^T\mathbf{y}\end{bmatrix}=\begin{bmatrix}\overline{\mathbf{R}}^T\overline{\mathbf{R}} & \overline{\mathbf{R}}^T\mathbf{g} & \overline{\mathbf{R}}^T\overline{\mathbf{r}}\\ \mathbf{g}^T\overline{\mathbf{R}} & \mathbf{g}^T\mathbf{g}+\gamma^2 & \mathbf{g}^T\overline{\mathbf{r}}+\gamma\overline{\rho}\\ \overline{\mathbf{r}}^T\overline{\mathbf{R}} & \overline{\mathbf{r}}^T\mathbf{g}+\gamma\overline{\rho} & \overline{\mathbf{r}}^T\overline{\mathbf{r}}+\overline{\rho}^2+\tau^2\end{bmatrix} \tag{3.32}$$

In combination with (3.28) the following relations can be derived. These relations are used to update the matrix $\mathbf{R}_a$:

$$\overline{\mathbf{R}}=\mathbf{R} \tag{3.33a}$$
$$\mathbf{g}=\mathbf{R}^{-T}\mathbf{X}^T\mathbf{x} \tag{3.33b}$$
$$\overline{\mathbf{r}}=\mathbf{r} \tag{3.33.c}$$
$$\gamma^2=\mathbf{x}^T\mathbf{x}-\mathbf{g}^T\mathbf{g} \tag{3.33d}$$
$$\overline{\rho}=\frac{1}{\gamma}(\mathbf{x}^T\mathbf{y}-\mathbf{g}^T\mathbf{r}) \tag{3.33e}$$
$$\tau^2=\mathbf{y}^T\mathbf{y}-\mathbf{r}^T\mathbf{r}-\overline{\rho}^2 \tag{3.33f}$$

In this way a key sample can be added to **X** and the corresponding **R** can be calculated without Cholesky decomposition on the ´updated´ **X**.

3.1.5 Stop Criterion

Including key samples to **X** will make the prediction more accurate. However, including all potential key samples in **X** is too demanding on computer resources and there would not be any data compression. The first key samples, which are included to **X** will largely improve the accuracy of the prediction, but every key sample included will result in a smaller improvement than its predecessor. Especially if the training data set is contaminated with noise, it could be that adding extra key samples will result in fitting noise into the approximation. Therefore, it is necessary to implement a criterion, that states when to stop the inclusion of key samples. 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 potential key sample. If the output values of the trainings set are contaminated with additive Gaussian noise with a variance of σ^2, then:

$$\frac{S^2}{\sigma^2} \sim \chi_1^2 \qquad \Leftrightarrow \qquad \mathbf{a}_i=0 \tag{3.34}$$

Here χ_1^2 denotes a chi-square distribution with one degree of freedom. A potential key sample $\mathbf{a}_i$ which contains no information, is denoted as "$\mathbf{a}_i$=0". If $\mathbf{a}_i$ is zero, than the probability that a certain error reduction is found is calculated as:

$$P(\frac{S^2}{\sigma^2} \geq z_\zeta \mid \mathbf{a}=0) < \zeta \tag{3.35}$$

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_1^2 distribution. The factor $\frac{S^2}{\sigma^2}$ is used as a measure for the error reduction of the approximation. So what is being calculated is the chance of having a certain error reduction given that the vector **a** is zero (contains no information). If this chance is very small it means that **a** is probably unequal to zero. If it would be equal to zero it would be very improbable that the given error reduction would occur. In Figure 3.9 this is explained graphically.

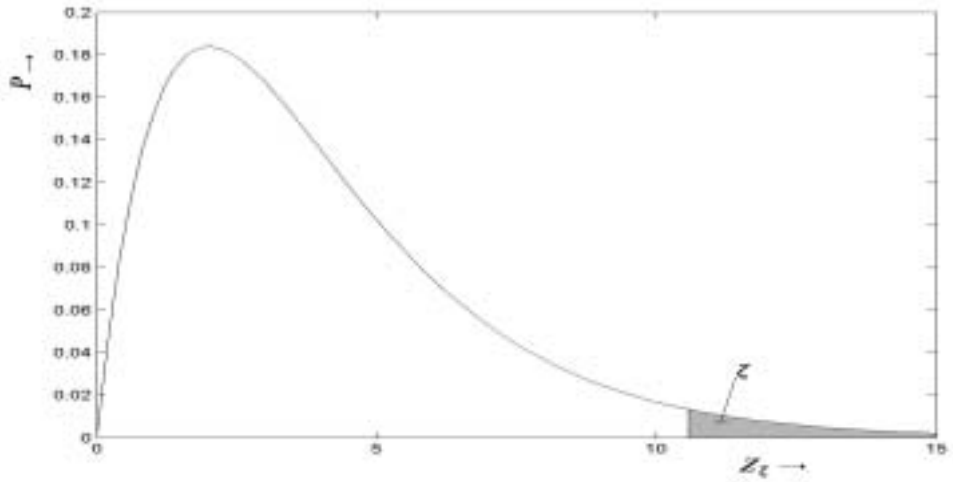

Figure 3.9. Chi-square distribution with one degree of freedom

In the figure above the horizontal-axis represents the value of $\frac{S^2}{\sigma^2}$. The chance of having this error reduction with given that **a** is zero, is equal to ζ (shown as the grey surface). This figure shows clearly that the larger the error reduction, the chances of vector **a** containing no information becomes smaller. If the chance of having a certain error reduction with $\boldsymbol{a}=0$ is very small then this new parameter is probably unequal to zero and should therefore be included as a key sample.

3.1.6 Approximate the function

When the function approximator has extracted the information from the training set, it has "learned" the target function. The approximator is able to give an output value of the function for any given input, which lies within the input space of the training set. The function output value is calculated as:

$$\hat{y}(\mathbf{x}_{new}) = \sum_{i=1}^{n} \tilde{f}_i(\mathbf{x}_{new}) b_i \tag{3.36}$$

Here b is calculated as in (2.11) and $\tilde{f}_i$ is the indicator function described by formula (3.6). It can be seen that the approximated output value is calculated summing the multiplications of each indicator function with its corresponding weight.

In Figure 3.10 a detailed block diagram of the offline KSM algorithm is shown. Here are all previously treated steps shown in relation to each other.

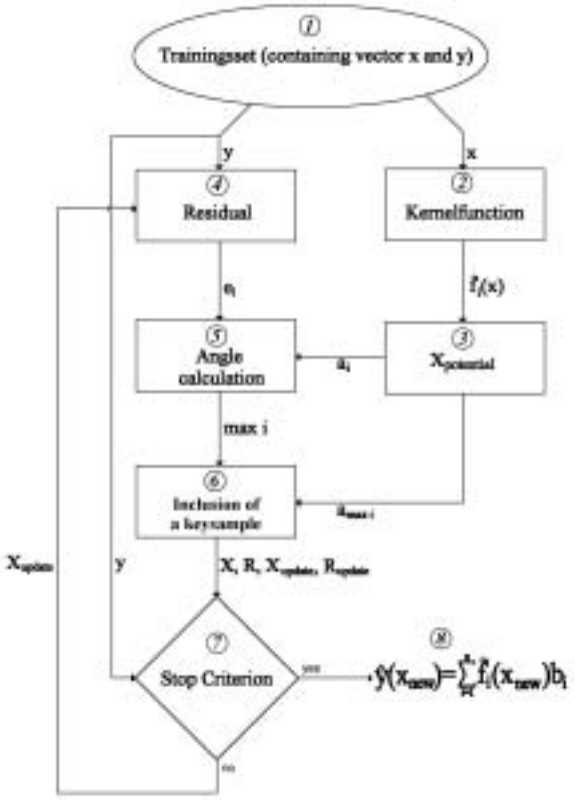

Figure 3.10. Detailed block diagram of offline KSM

3.1.7 Matlab Results

The offline algorithm described in this section has been implemented in Matlab. The Matlab code is included in Appendix B. The offline KSM is tested by approximating function (3.3). First a data set with 250 samples is generated and these are presented to the algorithm. The graphical output of the algorithm is shown in Figure 3.11. This approximation uses 23 key samples. Compared with the 250 samples in the original dataset it means there is a reduction of 90 % in the number of used samples.

In Figure 3.11 the dots are the samples in the training set, which are corrupted by Gaussian noise in their output values. The line represents the approximation made by the offline KSM. In the area where the derivative is relative small there is no or very little fitting of noise into the approximation. The area where the function has large derivative values the offline key sample machine is in contrast to the least squares approximators, capable of approximating the function rather well.

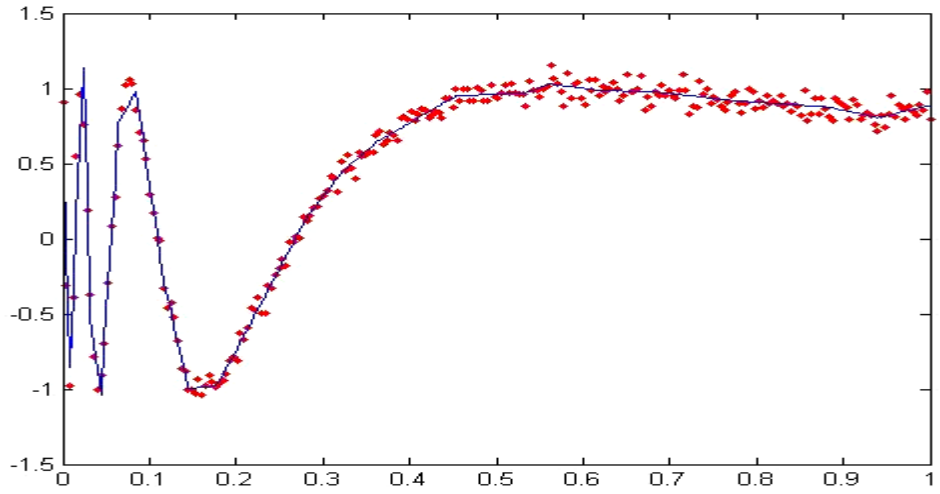

Figure 3.11. Graphical output of the offline key sample machine

3.2 Online Key Sample Machine

In section 3.1 it was shown that the offline KSM makes an approximation based on the information contained by all the samples in the training set. All samples are at the same time available to the algorithm. By selecting samples that contain most information as key samples, it is possible to approximate the function with a minimum of kernel functions. However, in practice it cannot always be guaranteed that all samples will be available beforehand. When the function approximation is done while the system that has to be controlled, is in operating mode, samples will become available one at a time. Storing these samples to construct a training set, which contains enough samples that are well distributed over input space, will take time. This especially will be the case if the system only works in certain areas of the input space for longer periods of time. During the period that the training set is being constructed the feed-forward controller cannot improve the performance of the system. This is a major drawback of the offline KSM algorithm. Therefore an online key sample machine was developed.

The online or Recursive Key Sample Machine (RKSM) approximates functions recursively. This means that RKSM is presented with one sample at a time and updates the approximation with the information contained by this sample. Therefore it is possible to make an approximation without having to wait for a training set with enough well distributed samples. The feed-forward controller can improve the performance of the controlled system from the moment it receives information about the system contained by incoming samples. Because of this continuous updating, RKSM is able to approximate even when the target function is somewhat dynamic because of changing system parameters (e.g. temperature). The disadvantage of recursive approximation is that when only few samples have been received, it is impossible to determine the amount of relative information contained by these samples. Therefore the assignment of key samples is especially in the beginning, non optimal. This however is later on corrected by evaluation key samples and removing those, which contain not enough information.

The RKSM algorithm can perform three approximation-updating actions. These are:

- Fine tuning the current approximation based on an incoming sample
- Extending the set of key samples with an incoming sample
- Omitting a key sample which contains not enough information

The updating actions will be treated here. An incoming sample is considered lying inside the uncertainty boundaries of the existing approximation. These boundaries are caused by measurement noise on the samples and consequently causing uncertainty in the approximation.

On the left side in Figure 3.12 an approximation based on three key samples and one incoming sample is shown. The information of this incoming sample is used to fine-tune the existing key samples. On the right side of figure the updated key samples and approximation is shown. The more a key sample is affected by this update the larger it is drawn. It contains more information about the target function and therefore its relative weight increases compared to the other key samples.

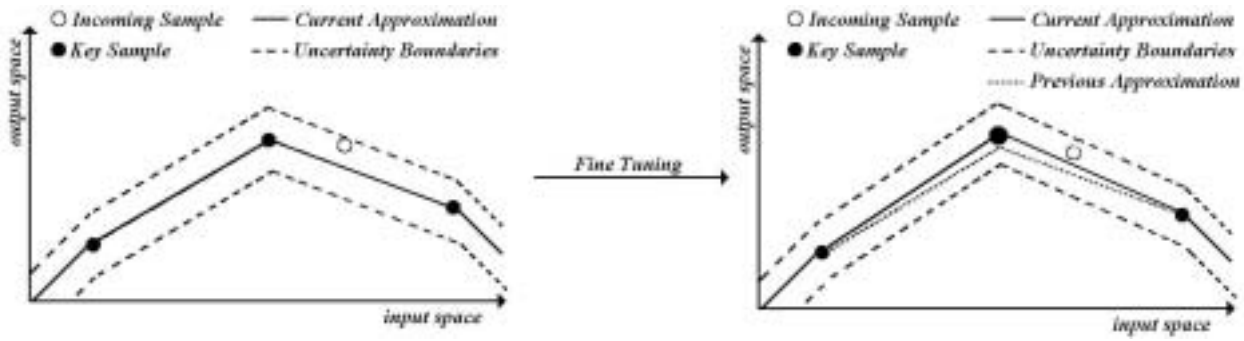

Figure 3.12. Incoming sample is used to fine tune key samples

When an incoming sample is lying outside the uncertainty boundaries fine-tuning the existing key samples does not suffice. A structural change has to be made by extending the set of key samples with the incoming sample. This is drawn in Figure 3.13.

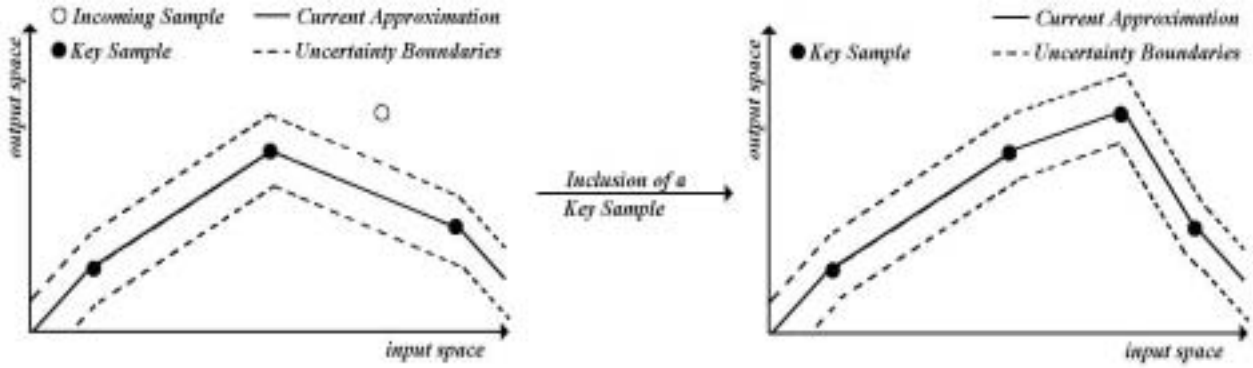

Figure 3.13. Inclusion of a key sample

Because there is no knowledge about future samples it is impossible to determine the amount of information an incoming sample contains. In coming samples can only be compared with previous samples. Therefore the first samples have a large probability of becoming key samples. It is possible that a sample that was included as a key sample, contained a relative large amount of information in the early stages of the approximation, but has a relative small contribution when more samples have been presented to the RKSM. The RKSM will evaluate the relative amount of information of each key sample and if the contribution of a certain key sample is to small, it will be omitted. If the approximation after omission lies within the uncertainty boundaries the key sample can be omitted. This is graphically explained in Figure 3.14.

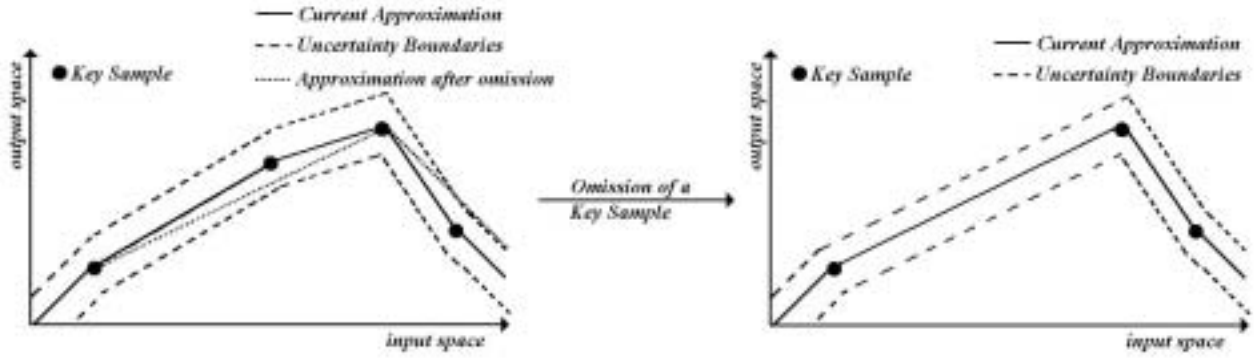

Figure 3.14. Omission of a key sample

The RKSM algorithm can perform the three-update actions described above. The algorithm itself will be treated in more detail in the rest of this section. The block diagram of the algorithm is shown below.

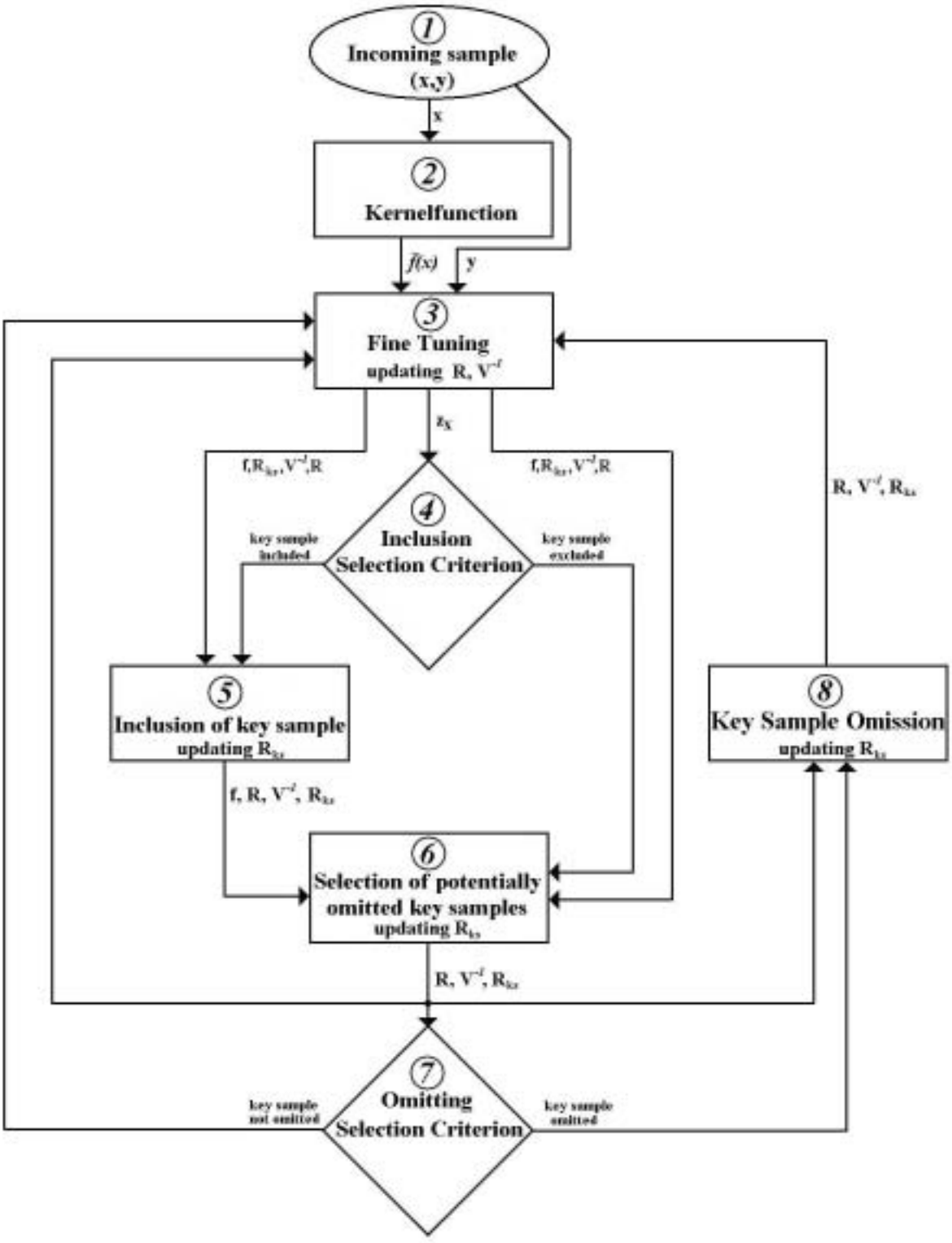

Figure 3.15. Block diagram of RKSM algorithm

The algorithm is recursive and the learning process continuous while the process is being controlled. Therefore there is no final step in the algorithm. When an output is needed the augmented matrix **R** can be used for determining the weight vector **b**. Calculating the output value of the target function is done in exactly the same way as explained in 3.1.6.

3.2.1 Incoming sample

The input for the RKSM consists of one sample at a time. The number of samples in the training set is not limited to certain values. RKSM will include information from incoming samples continuously, unless the user chooses to limit this "learning" period. As with KSM, for a good approximation it is important to have sufficient well-distributed samples.

3.2.2 Kernel function

The kernel functions, which have been treated in section 3.1, have the same purpose in RKSM as they have in KSM.

3.2.3 Fine-tuning

In the KSM algorithm all samples *N,* are used to create the indicator matrix $\mathbf{X}_{\text{potential}}$ [N x N]. The column vectors that carry the most information about the target function are selected and become key samples *n*, with which indicator matrix **X** [N x n] is constructed. It is impossible to implement this procedure in RKSM because of:

- The unlimited amount of samples in the training set and the corresponding infinitely large indicator matrix $\mathbf{X}_{\text{potential}}$ and **X**.
- Only samples that already have been presented to the algorithm are known, there is no information about future samples and therefore these samples cannot be used to construct $\mathbf{X}_{\text{potential}}$.

To circumvent this problem the following is done:

With the samples in the training set, two sets are constructed.

Reduced set: This set contains only the *n* key samples
Full set: This set contains all the training data

Matrix $\mathbf{X}_{\text{ks}}$ is the indicator matrix for the reduced set and is defined as in formula (3.37).

$$\mathbf{X}_{ks} = \begin{bmatrix} \tilde{f}_1(\mathbf{x}_1) & \tilde{f}_2(\mathbf{x}_1) & \cdots & \tilde{f}_n(\mathbf{x}_1) \\ \tilde{f}_1(\mathbf{x}_2) & \tilde{f}_2(\mathbf{x}_2) & \cdots & \tilde{f}_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_n) & \tilde{f}_2(\mathbf{x}_n) & \cdots & \tilde{f}_n(\mathbf{x}_{n)} \end{bmatrix} \tag{3.37}$$

The matrix $\mathbf{X}_{\text{ks}}$ has a dimension of [n x n] and contains the kernel function values for all key samples. Matrix **X,** which is also used in KSM is defined as:

$$\mathbf{X} = \begin{bmatrix} \tilde{f}_1(\mathbf{x}_1) & \tilde{f}_2(\mathbf{x}_1) & \cdots & \tilde{f}_n(\mathbf{x}_1) \\ \tilde{f}_1(\mathbf{x}_2) & \tilde{f}_2(\mathbf{x}_2) & \cdots & \tilde{f}_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_n) & \tilde{f}_2(\mathbf{x}_n) & \cdots & \tilde{f}_n(\mathbf{x}_n) \\ \tilde{f}_1(\mathbf{x}_{n+1}) & \tilde{f}_2(\mathbf{x}_{n+1}) & \cdots & \tilde{f}_n(\mathbf{x}_{n+1}) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{f}_1(\mathbf{x}_N) & \tilde{f}_2(\mathbf{x}_N) & \cdots & \tilde{f}_n(\mathbf{x}_N) \end{bmatrix} \left.\begin{matrix} \\ \\ \\ \\ \\ \\ \\ \end{matrix}\right\} \mathbf{X}_{\text{ks}} \tag{3.38}$$

In order to let the key samples represent the complete data set, a weight is introduced so that the normal and weighted normal equation become identical. This matrix $\mathbf{X}_{ks}$ is related to indicator matrix $\mathbf{X}$ [N x n] in the following way:

$$\mathbf{X}^T\mathbf{X} = \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{X}_{ks} \tag{3.39}$$

The non-weighted (3.11) and weighted normal equations are given in formula (3.40) and (3.41).

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y} \tag{3.40}$$

$$\mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{X}_{ks}\mathbf{b} = \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{y}_{ks} \tag{3.41}$$

Matrix $\mathbf{V}^{-1}$ [n x n] is the covariance matrix, which is used in weighting the key samples. The weight indicates how much information each key sample contains related to the other key samples. The non-weighted and weighted normal equations have to be identical. If the left-hand sides of the equations are made equal, it follows that:

$$\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{X}_{ks}\mathbf{b} \quad \Rightarrow \quad \mathbf{V}^{-1} = \mathbf{X}_{ks}^{-T}\mathbf{X}^T\mathbf{X}\mathbf{X}_{ks}^{-1} \tag{3.42}$$

The same for the right-hand side:

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{y}_{ks} \tag{3.43a}$$

$$\mathbf{y}_{ks} = \mathbf{V}\mathbf{X}_{ks}^{-T}\mathbf{X}^T\mathbf{y} \tag{3.43b}$$

$$= \mathbf{X}_{ks}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}_{ks}^{-T}\mathbf{X}_{ks}^{-T}\mathbf{X}^T\mathbf{y} \tag{3.43c}$$

$$= \mathbf{X}_{ks}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.43d}$$

This means that the output of the key samples $\mathbf{y}_{ks}$ is equal to the prediction of the full data estimator at the key samples. The prediction of the reduced approximator is given as:

$$\mathbf{y}_{ks}(\mathbf{x}_{ks}) = \mathbf{X}_{ks}\mathbf{b} \tag{3.44}$$

Substituting (3.41) for $\mathbf{b}$ into this equation gives:

$$\mathbf{y}_{ks}(\mathbf{x}_{ks}) = \mathbf{X}_{ks}(\mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{X}_{ks})^{-1}\mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{y}_{ks} \tag{3.45a}$$

$$= \mathbf{X}_{ks}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}_{ks}^T\mathbf{X}_{ks}^{-T}\mathbf{X}^T\mathbf{X}\mathbf{X}_{ks}^{-1}\mathbf{y}_{ks} = \mathbf{y}_{ks} \tag{3.45b}$$

From this it can be concluded that the predictions of the full data approximator (3.43d) and the reduced approximator (3.45b) are equal.

For computing convenience $\mathbf{X}$ and $\mathbf{X}_{ks}$ are QR-decomposed.

$$\mathbf{X} = \mathbf{Q}\mathbf{R} \tag{3.46}$$

$$\mathbf{X}_{ks} = \mathbf{Q}_{ks}\mathbf{R}_{ks} \tag{3.47}$$

Formula (3.39) can be written as:

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}_{ks}\mathbf{R}_{ks}^T\mathbf{V}^{-1}\mathbf{R}_{ks}^T\mathbf{R}_{ks} \tag{3.48}$$

So covariance matrix $\mathbf{V}^{-1}$ is given as:

$$\mathbf{V}^{-1} = \mathbf{R}_{ks}^{T}\mathbf{R}_{ks}^{-T}\mathbf{R}^{T}\mathbf{R}\mathbf{R}_{ks}^{-T}\mathbf{R}_{ks}^{T} \tag{3.49}$$

The algorithm uses and stores only the matrices **R,** $\mathbf{R}_{ks}$ and $\mathbf{V}^{-1}$. If a sample is presented to the RKSM algorithm the input space information is projected into feature space, by the kernel functions. This information is used to fine-tune the existing key samples. As the number of key samples does not alter, the matrices **X** and $\mathbf{R}_{ks}$ remain unchanged. The augmented indicator matrix is extended by a row due to the new training sample.

$$\mathbf{X}_a = \begin{bmatrix}\mathbf{X} & \mathbf{y}\end{bmatrix} \Rightarrow \mathbf{X}_a = \begin{bmatrix}\mathbf{X} & \mathbf{y} \\ \mathbf{f} & y\end{bmatrix} \tag{3.50}$$

In which y is the output value of the incoming sample and **f** is a vector that contains the output of the kernel functions for the x-value of the incoming sample. Instead of using $\mathbf{X}_a$ the algorithm uses the matrix $\mathbf{R}_a$ from the QR-decomposition of $\mathbf{X}_a$. Matrix $\mathbf{R}_a$ is stored in its augmented form as it is done in KSM. The information of the incoming sample will change the augmented **R** matrix in the following way:

$$\mathbf{R}_a = \begin{bmatrix}\mathbf{R} & \mathbf{z}_\mathbf{X}\end{bmatrix} \Rightarrow \mathbf{R}_a = \begin{bmatrix}\mathbf{R} & \mathbf{z}_\mathbf{X} \\ \mathbf{f} & y\end{bmatrix} \tag{3.51}$$

By means of a Givens rotation, this matrix is reduced in the following form:

$$\mathbf{R}_a = \begin{bmatrix}\mathbf{R} & \mathbf{z}_\mathbf{X} \\ \mathbf{f} & y\end{bmatrix} \Rightarrow \mathbf{R}_a = \begin{bmatrix}\overline{\mathbf{R}} & \overline{\mathbf{z}}_\mathbf{X} \\ \mathbf{0} & \varsigma\end{bmatrix} \tag{3.52}$$

In which $\overline{\mathbf{R}}$ is upper-triangular and ζ^2 is the change in summed squared error due to the new sample. The Givens rotation algorithm calculates an angle by which the orthonormal basis spanned by **Q** is rotated. Matrix **R** contains the projections of **X** on this basis of **Q**. This rotation is done in such a way that matrix **R** becomes upper-triangular again. The Givens rotation is explained in detail in appendix A.

The weight matrix $\mathbf{V}^{-1}$ relates the matrix **R** and $\mathbf{R}_{ks}$ as shown in (3.48). The weights indicate the number of samples each key sample represents. Therefore the weights have to be updated when the key samples have been fine-tuned. The updated matrix $\mathbf{V}^{-1}$ can of course be calculated with (3.48) but it demands less computing resources to update the weight matrix as follows:

$$\mathbf{u} = \overline{\mathbf{R}}_{ks}^{-1}\overline{\mathbf{R}}_{ks}^{-T}\mathbf{f}^{T} \tag{3.53a}$$

$$\overline{\mathbf{V}}^{-1} = \mathbf{V}^{-1} + \mathbf{u}^{T}\mathbf{u} \tag{3.53b}$$

In (3.53) $\overline{\mathbf{R}}_{ks}$ is the fine-tuned matrix $\mathbf{R}_{ks}$ and $\overline{\mathbf{V}}^{-1}$ is the updated weight matrix.

3.2.4 Inclusion selection criterion

The fine-tuning incorporates information from the incoming sample into the existing key samples. However, it is possible that the sample contains information about the target function that cannot be represented by the existing key samples. In that case the fine-tuning is does not suffice and the set of key samples has to be extended with the incoming sample. The decision whether a sample has to be included into the key sample set made based on the value of ζ^2 after fine-tuning the key samples. If ζ^2 the summed squared error, is larger than a certain value it means that the incoming sample contains a relatively large amount of information that cannot be presented by the fine-tuned key

samples. In that case the sample will be included as will be described in paragraph 3.2.5. When the summed squared error is relatively small the RKSM algorithm will proceed with selecting potentially omitted key samples, which is treated in 3.2.6.

3.2.5 Inclusion of a key sample

A sample that has to be added to the set of key samples will affect matrices $\mathbf{R}$, $\mathbf{R}_{ks}$ and $\mathbf{V}^{-1}$. The weight matrix $\mathbf{V}^{-1}$ will be extended with one row and one column, as done in (3.54).

$$\overline{\mathbf{V}}^{-1} = \begin{bmatrix} \mathbf{V}^{-1} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.54}$$

Here $\overline{\mathbf{V}}^{-1}$ is the updated weight matrix. Because the samples that were presented previously to the RKSM were not stored, it is impossible to determine the covariance between these samples and the current incoming sample. Therefore an assumption is made that the incoming sample is uncorrelated with the previous samples. Due to this assumption, the update is not exact.

After the update the relation given in (3.48) which is repeated here for convenience, still has to hold.

$$\overline{\mathbf{R}}^T\overline{\mathbf{R}} = \overline{\mathbf{R}}_{ks}\overline{\mathbf{R}}_{ks}^T\overline{\mathbf{V}}^{-1}\overline{\mathbf{R}}_{ks}^T\overline{\mathbf{R}}_{ks} \tag{3.55}$$

The bar denotes matrices that have been updated by the inclusion of a key sample. For simplicity and clarity reasons this relation is written as:

$$\overline{\mathbf{R}}_a^T\overline{\mathbf{R}}_a = \overline{\mathbf{X}}_{ks,a}^T\overline{\mathbf{V}}^{-1}\overline{\mathbf{X}}_{ks,a} \tag{3.56}$$

in which $\overline{\mathbf{R}}_a$ is as in (3.30) and

$$\overline{\mathbf{X}}_{ks,a} = \begin{bmatrix} \mathbf{X}_{ks} & \mathbf{f}^T & \mathbf{y} \\ \mathbf{f} & \phi & y \end{bmatrix} \tag{3.57}$$

The subscript a is used to indicate that the matrix is used in its augmented form. Variable $\phi = \tilde{f}_{k+1}(x_{k+1})$ is the value of the new dual indicator function for the new key sample. Writing out (3.56) gives:

$$\begin{bmatrix} \overline{\mathbf{R}} & \mathbf{g} & \overline{\mathbf{r}} \\ 0 & \gamma & \overline{\rho} \\ 0 & 0 & \tau \end{bmatrix}^T \left[\begin{bmatrix} \overline{\mathbf{R}} & \mathbf{g} & \overline{\mathbf{r}} \\ 0 & \gamma & \overline{\rho} \\ 0 & 0 & \tau \end{bmatrix} \right] = \begin{bmatrix} \mathbf{X}_{ks,a} & \mathbf{f}^T & \mathbf{y} \\ \mathbf{f} & \phi & y \end{bmatrix}^T \begin{bmatrix} \mathbf{V}^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{ks,a} & \mathbf{f}^T & \mathbf{y} \\ \mathbf{f} & \phi & y \end{bmatrix} \tag{3.58a}$$

$$\begin{bmatrix} \overline{\mathbf{R}}^T\overline{\mathbf{R}} & \overline{\mathbf{R}}^T\mathbf{g} & \overline{\mathbf{R}}^T\overline{\mathbf{r}} \\ \mathbf{g}^T\overline{\mathbf{R}} & \mathbf{g}^T\mathbf{g} + \gamma^2 & \mathbf{g}^T\overline{\mathbf{r}} + \gamma\overline{\rho} \\ \overline{\mathbf{r}}^T\overline{\mathbf{R}} & \overline{\mathbf{r}}^T\mathbf{g} + \gamma\overline{\rho} & \overline{\mathbf{r}}^T\mathbf{r} + \overline{\rho}^2 + \tau^2 \end{bmatrix} =$$

$$\begin{bmatrix} \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{X}_{ks}+\mathbf{f}^T\mathbf{f} & \mathbf{X}_{ks}\mathbf{V}^{-1}\mathbf{f}^T+\mathbf{f}^T\phi & \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{y}+\mathbf{f}^T y \\ \mathbf{f}\mathbf{V}^{-1}\mathbf{X}_{ks}+\phi\mathbf{f} & \mathbf{f}\mathbf{V}^{-1}\mathbf{f}^T+\phi^2 & \mathbf{f}\mathbf{V}^{-1}\mathbf{y}+\phi y \\ \mathbf{y}^T\mathbf{V}^{-1}\mathbf{X}_{ks}+y\mathbf{f} & \mathbf{y}^T\mathbf{V}^{-1}\mathbf{f}^T+\phi y & \mathbf{y}^T\mathbf{V}^{-1}\mathbf{y}+y^2 \end{bmatrix} \tag{3.58b}$$

From this all the equations necessary for the update can be derived. This procedure implies adding a row and a column to $\overline{\mathbf{R}}_a$ and $\overline{\mathbf{X}}_{ks,a}$. However, fine-tuning **R**, is equal to adding only a row. As can be seen from the relation below:

$$\begin{bmatrix} \overline{\mathbf{R}} & \overline{\mathbf{r}} \\ 0 & \overline{\rho} \end{bmatrix}^T \begin{bmatrix} \overline{\mathbf{R}} & \overline{\mathbf{r}} \\ 0 & \overline{\rho} \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{ks} & \mathbf{y} \\ \mathbf{f} & y \end{bmatrix}^T \begin{bmatrix} \mathbf{V}^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{X}_{ks} & \mathbf{y} \\ \mathbf{f} & y \end{bmatrix} \tag{3.59a}$$

$$\begin{bmatrix} \overline{\mathbf{R}}^T\overline{\mathbf{R}} & \overline{\mathbf{R}}^T\overline{\mathbf{r}} \\ \overline{\mathbf{r}}^T\overline{\mathbf{R}} & \overline{\mathbf{r}}^T\overline{\mathbf{r}}+\rho^2 \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{X}_{ks}+\mathbf{f}^T\mathbf{f} & \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{y}+\mathbf{f}^T y \\ \mathbf{y}^T\mathbf{V}^{-1}\mathbf{X}_{ks}+y\mathbf{f} & \mathbf{y}^T\mathbf{V}^{-1}\mathbf{y}+y^2 \end{bmatrix} \tag{3.59b}$$

It can be seen that calculating $\overline{\mathbf{R}}$ and $\overline{\mathbf{r}}$ from (3.58b) and (3.59b) gives exactly the same result. So it can be concluded that by fine-tuning $\mathbf{R}_a$ as done in (3.52), $\overline{\mathbf{R}}$ and $\overline{\mathbf{r}}$ are automatically calculated. The $\overline{\mathbf{r}}$ used in (3.58) and (3.59) equals the $\overline{\mathbf{z}}_{\mathbf{X}}$ in (3.52). The remaining vector **g** and parameters γ and ρ can be calculated from the equation in (3.58b).

$$\mathbf{g} = \overline{\mathbf{R}}^{-T}\mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{f}^T+\mathbf{f}^T\phi \tag{3.60a}$$

$$\gamma = \sqrt{\mathbf{f}\mathbf{V}^{-1}\mathbf{f}^T+\phi^2-\mathbf{g}^T\mathbf{g}} \tag{3.60b}$$

$$\overline{\rho} = \frac{1}{\gamma}(\mathbf{f}\mathbf{V}^{-1}\mathbf{y}+\phi y-\mathbf{g}^T\overline{\mathbf{r}}) \tag{3.60c}$$

Variable τ is not calculated, because it has no influence on the prediction. Because $\mathbf{X}_{ks}$ itself is not used or stored in the RKSM the relation

$$\mathbf{R}_{ks}^T\mathbf{R}_{ks} = \mathbf{X}_{ks} \tag{3.61}$$

is used. The matrix $\mathbf{R}_{ks}$ itself has to be updated too and the relation above still has to hold after the update. In (3.62) this relation is written out for after the update.

$$\begin{bmatrix} \mathbf{R}_{ks} & \mathbf{r} \\ 0 & \rho \end{bmatrix}^T \begin{bmatrix} \mathbf{R}_{ks} & \mathbf{r} \\ 0 & \rho \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{ks} & \mathbf{f}^T \\ \mathbf{f} & \phi \end{bmatrix} \tag{3.62a}$$

$$\begin{bmatrix} \mathbf{R}_{ks}^T\mathbf{R}_{ks} & \mathbf{R}_{ks}^T\mathbf{r} \\ \mathbf{r}^T\mathbf{R}_{ks} & \mathbf{r}^T\mathbf{r}+\rho^2 \end{bmatrix} = \begin{bmatrix} \mathbf{X}_{ks} & \mathbf{f}^T \\ \mathbf{f} & \phi \end{bmatrix} \tag{3.62b}$$

This relation gives the equations for updating $\mathbf{R}_{ks}$.

$$\mathbf{r} = \mathbf{R}_{ks}^{-T}\mathbf{f}^T \tag{3.63a}$$

$$\rho = \sqrt{\phi-\mathbf{r}^T\mathbf{r}} \tag{3.63b}$$

Variable ρ can be used to test whether the new key sample contains enough information, which will be discussed in 3.1.5.

3.2.6 Selection of potentially omitted key samples

By adding key samples as done in 3.2.5, the contribution of improving the approximation by existing key samples changes. It is possible that the contribution of some key samples becomes so small that omitting them will have little effect on the accuracy of the approximation. The increase in error by omitting a key sample can be calculated but takes considerable computation time. Therefore the error increase is only calculated for the omission of the key sample that contains the smallest amount of information. Selecting this key sample is done with relation (3.64), which is taken from (De Kruif, 2005).

$$\Delta E = \frac{b_i^2}{4[(\mathbf{X}^T\mathbf{X})^{-1}]_{ii}} \tag{3.64}$$

In which ΔE is the change in the approximation error, **b** is the parameter vector and **X** represents the indicator matrix. The term $\sigma^2(\mathbf{X}^T\mathbf{X})^{-1}$ is the covariance matrix of the parameter vector **b**. So the change in the approximation error due to omission of a key sample is determined by the value of the corresponding element in the parameter vector and the variance of this element. The parameter value divided by its uncertainty is proportional to the test whether the hypothesis "b=0" can be rejected. This hypothesis is rejected with a significance ζ if:

$$P\left(\frac{b_i^2}{\sigma^2[(\mathbf{X}^T\mathbf{X})^{-1}]_{ii}} > z_\zeta\right) < \zeta \tag{3.65}$$

In which z_ζ and ζ are coupled by a χ_1^2 distribution. From this relation it can be seen that the key sample, which is most likely to be zero, will give the smallest increase of error. In (3.65) the parameter value is divided by the uncertainty of this parameter. Another possibility selecting a candidate for omission is to divide the parameter by the uncertainty of the key sample. This results in:

$$\textit{index candidate key sample} = \arg_i \min \frac{b_i^2}{[\mathbf{V}]_{ii}} = \arg_i \min b_i^2 \left[\mathbf{V}^{-1}\right]_{ii},\ I=1\ldots n \tag{3.66}$$

The ratio that is used in this expression is not directly related to the change in error given in (3.64). It is however suitable for selecting the omission candidate key sample, because it selects the key sample that has a small influence and represents a small number of samples.
Using $\mathbf{V}^{-1}$ instead of $(\mathbf{X}^T\mathbf{X})^{-1}$ has another advantage because calculating the latter will cause numerical instability if key samples are much alike. Values on the diagonal of $\mathbf{V}^{-1}$, indicate how many samples are represented by a key sample and therefore it is unlikely that these values will cause numerical instability.

3.2.7 Omission selection criterion

When the least contributing key sample is selected, it has to be evaluated whether it can be omitted without causing a large increase of the approximation error. If the omission candidate is denoted by the column vector **p**, matrix $\mathbf{R}_a$ can be partitioned in the following way

$$\mathbf{R}_a = \left[\mathbf{R}_1 \quad \mathbf{p} \quad \mathbf{R}_2 \quad | \quad \mathbf{z}_\mathbf{X}\right] \tag{3.67}$$

The columns of $\mathbf{R}_a$ are changed into:

$$\mathbf{R}_a = \begin{bmatrix} \mathbf{R}_1 & \mathbf{R}_2 & \mathbf{p} & | & \mathbf{z}_\mathbf{X} \end{bmatrix} \tag{3.68}$$

This will make matrix $\mathbf{R}_a$ no longer upper-triangular. By means of a Givens rotation (Appendix A), which has been described in *3.2.3*, $\mathbf{R}_a$ can be brought back into its upper-triangular shape:

$$\overline{\mathbf{R}}_a = \begin{bmatrix} \overline{\mathbf{R}}_1 & \overline{\mathbf{R}}_2 & \overline{\mathbf{p}} & | & \overline{\mathbf{z}}_\mathbf{X} \end{bmatrix} \tag{3.69}$$

The last element of $\overline{\mathbf{z}}_\mathbf{X}$ is the weighted error increase due to the removal of the key sample, which is represented by $\overline{\mathbf{p}}$. If the value of this element is larger than specified in the omission criterion, the candidate key sample should not be removed. Once decided that a key sample will be omitted, it is simply a matter of removing $\overline{\mathbf{p}}$ and the last row from $\overline{\mathbf{R}}_a$. Hereafter, the procedure of shifting the selected column to the end of the matrix, applying Givens rotation and then removing the last row and column, is applied onto $\mathbf{R}_{ks}$. The first step of updating covariance matrix $\mathbf{V}^{-1}$ is set by rearranging the matrix in such a way that the column and row corresponding to the selected key sample become the last row and column. This is done in the following way:

$\mathbf{V}^{-1} =$

$\mathbf{V}_1^{-1}$	$\mathbf{I}_{c1}$	$\mathbf{V}_2^{-1}$	$\mathbf{E}_{c1}$
$\mathbf{I}_{r1}$	$\mathbf{I}_{c2}$	$\mathbf{I}_{r2}$	$\mathbf{E}_{c2}$
$\mathbf{V}_3^{-1}$	$\mathbf{I}_{c3}$	$\mathbf{V}_4^{-1}$	$\mathbf{E}_{c3}$
$\mathbf{E}_{r1}$	$\mathbf{I}_{c4}$	$\mathbf{E}_{r2}$	$\mathbf{E}_{c4}$

(3.70a)

$\overline{\mathbf{V}}^{-1} =$

$\mathbf{V}_1^{-1}$	$\mathbf{E}_{c1}$	$\mathbf{V}_2^{-1}$	$\mathbf{I}_{c1}$
$\mathbf{E}_{c1}^T$	$\mathbf{E}_{c4}^T$	$\mathbf{E}_{c3}^T$	$\mathbf{E}_{c2}^T$
$\mathbf{V}_3^{-1}$	$\mathbf{E}_{c3}$	$\mathbf{V}_4^{-1}$	$\mathbf{I}_{c3}$
$\mathbf{I}_{c1}^T$	$\mathbf{I}_{c4}^T$	$\mathbf{I}_{c3}^T$	$\mathbf{I}_{c2}^T$

(3.70b)

In which $\overline{\mathbf{V}}^{-1}$ is the rearranged covariance matrix. In (3.70a) $\mathbf{I}_{c1}$ stands for “index column 1” and $\mathbf{E}_{r1}$ stands for “End row 1”. After the rearrangement in which the column and row corresponding to the index of the selected key sample are shifted to the outer sides of the matrix, the actual omission can be executed. Before the omission (3.39), which is repeated here, has to hold:

$$\mathbf{X}^T\mathbf{X} = \mathbf{X}_{ks}^T\mathbf{V}^{-1}\mathbf{X}_{ks} \tag{3.71}$$

If these matrices are written out in a partitioned form, it gives:

$$\mathbf{X} = \begin{bmatrix} \overline{\mathbf{X}} & \mathbf{z} \end{bmatrix},\ \mathbf{X}_{ks} = \begin{bmatrix} \overline{\mathbf{X}}_{ks} & \mathbf{f}^T \\ \mathbf{f} & \phi \end{bmatrix},\ \mathbf{V}^{-1} = \begin{bmatrix} \mathbf{W}^{-1} & \mathbf{v} \\ \mathbf{v}^T & v^{-1} \end{bmatrix} \tag{3.72}$$

Here $\overline{\mathbf{X}}$ and $\overline{\mathbf{X}}_{ks}$ are the indicator matrices after the removal of a key sample and $\mathbf{V}^{-1}$ is the rearranged covariance matrix given in (3.70b). The remaining elements of these matrices do not

alter when a key sample is omitted. However the weights of the remaining key samples do alter and therefore it is not possible to find the updated weight matrix by partitioning it. It can be calculated because (3.71) should also hold for the updated matrices.

$$\overline{\mathbf{X}}^T\overline{\mathbf{X}} = \overline{\mathbf{X}}_{ks}^T\overline{\mathbf{V}}^{-1}\overline{\mathbf{X}}_{ks} \tag{3.73}$$

By substituting (3.72) into (3.71), it can be seen that:

$$\overline{\mathbf{X}}^T\overline{\mathbf{X}} = \overline{\mathbf{X}}_{ks}^T\mathbf{W}^{-1}\overline{\mathbf{X}} + \mathbf{f}^T\mathbf{v}^T\overline{\mathbf{X}}_{ks} + \overline{\mathbf{X}}_{ks}^T\mathbf{v}\mathbf{f} + \mathbf{f}^T v^{=1}\mathbf{f} \tag{3.74}$$

Formula (3.73) and (3.74) give the following relation:

$$\overline{\mathbf{X}}_{ks}^T\overline{\mathbf{V}}^{-1}\overline{\mathbf{X}}_{ks} = \overline{\mathbf{X}}_{ks}^T\mathbf{W}^{-1}\overline{\mathbf{X}} + \mathbf{f}^T\mathbf{v}^T\overline{\mathbf{X}}_{ks} + \overline{\mathbf{X}}_{ks}^T\mathbf{v}\mathbf{f} + \mathbf{f}^T v^{=1}\mathbf{f} \tag{3.75a}$$

$$\overline{\mathbf{V}}^{-1} = \mathbf{W}^{=1} + \overline{\mathbf{X}}_{ks}^{-T}\mathbf{f}^T\mathbf{v}^T + \mathbf{v}\mathbf{f}\overline{\mathbf{X}}_{ks}^{-1} + \overline{\mathbf{X}}_{ks}^{-T}\mathbf{f}^T v^{-1}\mathbf{f}\mathbf{X}_{ks}^{-1} \tag{3.75b}$$

All involved matrices have been updated and the selected key sample has been omitted. In practice it is not necessary to check whether key samples can be omitted every time the algorithm runs through the loop. It suffices to evaluate existing key samples after 5 tot 10 incoming samples, which saves a lot of computation time.

4 Blocked Key Sample Machine

In chapter 3 the offline and online key sample machine were explained in detail. In the offline variant all samples of the training set are presented to the algorithm at once. In contrast to the online variant, in which samples are presented one at a time. In this chapter a newly developed key sample machine is treated, called “Blocked Key Sample Machine”. This key sample machine is able to handle samples in variable block sizes. It works recursively, just as the online algorithm, but it can however handle more samples at a time. The handling of multiple samples is very similar to that of the offline key sample machine. By setting the block size equal to the number in the training set the blocked key sample machine will behave as the offline algorithm. A block size of one means its behavior will be like the online key sample machine. Therefore, it can be concluded that the offline and online algorithms are to extreme variants of the blocked key sample machine (BKSM). The blocked algorithm is recursive it can learn the function while the system is being controlled. However it can handle much more samples than the online key sample machine, because the latter has to fine tune and include incoming samples in a very time consuming way. The BKSM is able to select incoming samples, which contain a relative large amount of new information about the function and use these samples for updating the set of key samples. The less significant samples are only used for calculating the weights of the indicators. In Figure 4.1 a simple block diagram of the BKSM is drawn.

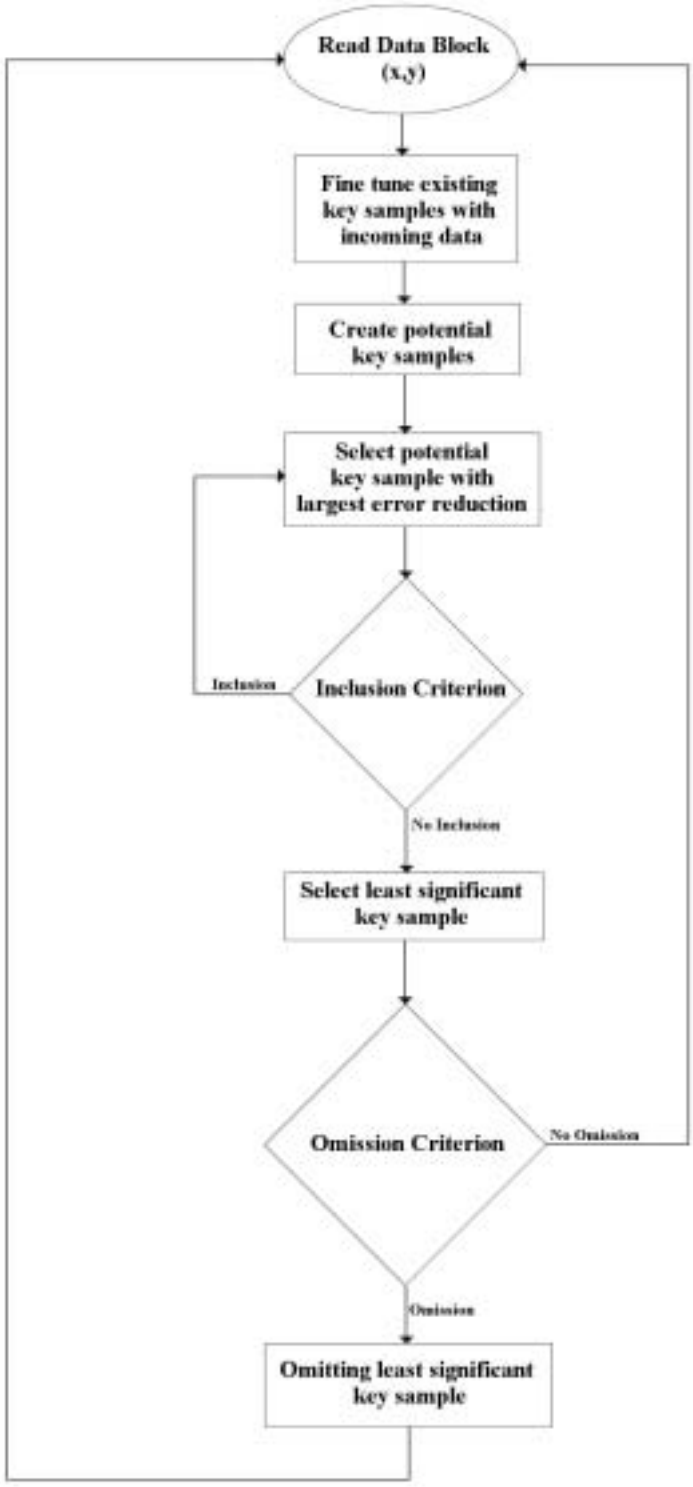

Figure 4.1. Simple block diagram of the blocked key sample machine

The block diagram in the figure above is almost similar to that of Figure 3.3. The differences are that the training set is presented in blocks to the algorithm and that the algorithm is recursive. Whenever an approximation is needed for control purposes it can be calculated, but there is no part in the algorithm were the calculations are finished and a final approximation is given, which is the case in the offline key sample machine. The blocked key sample machine will be treated in detail hereafter. Knowledge about the mathematical operations used in the offline and online algorithms is assumed.

4.1 Training set

The training set is divided into blocks with a fixed block size. This block size can vary between one sample and all samples in the training set. There is probably an optimum block size for calculation speed. Using a relative large block size has the advantage that samples can be compared on the relative amount of information they contain. This means that less significant samples will not be used in time-consuming parts of the algorithm. However using a very large block size will result in more memory-demanding computations and this can slow down the learning process.

4.2 Fine-Tuning

The incoming samples are used for fine-tuning the existing key samples. The existing set of key samples is stored in indicator matrix **X.** The key samples stored in this matrix each has their own weight, which indicates the number of sample the key sample represents. Matrix **X** can be written as:

$$\mathbf{X} = \left[\mathbf{P}^{-1}\mathbf{X}_{ks}\right] \tag{4.1}$$

In which **P** is related to, the in chapter 3.2 introduced covariance matrix $\mathbf{V}^{-1}$ in the following way:

$$\mathbf{P}^{T}\mathbf{P}=\mathbf{V}^{-1} \tag{4.2}$$

The incoming batch of samples is used to update indicator matrix **X**. Because each new sample represents only itself, the weight assigned to these samples equals one. The updated indicator matrix has the form:

$$\overline{\mathbf{X}} = \begin{bmatrix} \mathbf{P}^{-1}\mathbf{X}_{ks} \\ \mathbf{X}_{new} \end{bmatrix} \tag{4.3}$$

Here $\mathbf{X}_{\text{new}}$ contains the incoming samples projected in feature space by the existing key samples. As in the offline and online KSM algorithms the indicator matrix is augmented with the output values of the samples. Therefore the augmented indicator matrix $\overline{\mathbf{X}}_a$ can be written as:

$$\overline{\mathbf{X}}_a = \left[\overline{\mathbf{X}} \quad \mathbf{y}\right] = \begin{bmatrix} \mathbf{P}^{-1}\mathbf{X}_{ks} & \mathbf{P}^{-1}\mathbf{y}_{ks} \\ \mathbf{X}_{new} & \mathbf{y}_{new} \end{bmatrix} \tag{4.4}$$

The output values of the key samples and incoming samples are denoted respectively by $\mathbf{y}_{\text{ks}}$ and $\mathbf{y}_{\text{new}}$. For numerical stability and computational speed this indicator matrix is QR-decomposed. The decomposition is done in the same way as described in chapter 3.1. The augmented matrix $\mathbf{R}_{\text{a}}$ can be used for determining the weight vector **b** as shown in relation (3.19).

4.3 Selecting potential key samples

In 4.2 the incoming block of samples is used for fine-tuning the existing set of key samples. However, some samples contain information, which cannot be “learned” by this set of key samples. Therefore it is necessary to asses all incoming samples and determine the amount of error reduction that is obtained by adding them to the existing set op key samples. As in the offline algorithm the residual of the approximation is needed for this assessment. The residual is calculated by projecting the key sample and sample output values into feature space and then mapping it back to the output

space. The residual is the difference between the real output and the approximated output values. The mapping into feature space is done by relation (3.18), which is repeated here for convenience.

$$\mathbf{z}_{\mathbf{x}} = \mathbf{Q}^T \mathbf{y} \tag{4.5}$$

The residual is then calculated by:

$$\mathbf{e} = \mathbf{y} - \mathbf{Q}\mathbf{z}_{\mathbf{x}} \tag{4.6}$$

In the two relations above matrix **Q** is used for calculating purposes. In the offline and online algorithm this matrix was, if possible not calculated and certainly not stored. The reasons for using this matrix in the blocked algorithm are:

- The residual **e**, can be calculated without first calculating weight vector **b** as is done in the offline algorithm shown in formula (3.9).
- The change of the residual due to adding key samples can easily be calculated. This will be explained in more detail in section 4.4.
- In the online algorithm matrix **R** is updated with every incoming sample. In the blocked key sample machine multiple samples are presented simultaneously and if the online approach would be used it would mean that **R** has to be updated for every incoming sample. It is faster to add the samples to indicator matrix **X** as done in (4.3) and (4.4) and than calculate the updated matrix **R** by applying QR-decomposition on **X**. This QR-decomposition has as result that the matrix **Q** will be calculated anyway.
- Storing **Q** is equally demanding on memory requirements as storing **X**.

All incoming samples will be projected into feature space and these projections are stored in matrix $\mathbf{X}_{\text{potential}}$. When there are n key samples and N incoming samples, $\mathbf{X}_{\text{potential}}$ will have the form:

$$\mathbf{X}_{potential} = \begin{bmatrix} f_1(\mathbf{x}_{ks,1}) & f_2(\mathbf{x}_{ks,1}) & \cdots & f_N(\mathbf{x}_{ks,1}) \\ f_1(\mathbf{x}_{ks,2}) & f_2(\mathbf{x}_{ks,2}) & \cdots & f_N(\mathbf{x}_{ks,2}) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_{ks,n}) & f_2(\mathbf{x}_{ks,n}) & \cdots & f_N(\mathbf{x}_{ks,n}) \\ f_1(\mathbf{x}_{new,1}) & f_2(\mathbf{x}_{new,1}) & \cdots & f_N(\mathbf{x}_{new,1}) \\ f_1(\mathbf{x}_{new,2}) & f_2(\mathbf{x}_{new,2}) & \cdots & f_N(\mathbf{x}_{new,2}) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(\mathbf{x}_{new,N}) & f_2(\mathbf{x}_{new,N}) & \cdots & f_N(\mathbf{x}_{new,N}) \end{bmatrix} \tag{4.7}$$

The column vectors in this matrix represent the incoming samples in feature space and are the potential indicators. By calculating the angle between the residual and the column vectors it can be determined which incoming sample contains the most information. This is done with formula (3.10) in paragraph 3.1.2.2.

4.4 Inclusion of a key sample

In 4.3 the best potential key sample is selected. Calculations must be made to be certain whether it is useful to extend the set of key samples with this new indicator. First the new indicator is projected on the space spanned by the present key samples.

$$\mathbf{r}_{new} = \mathbf{Q}^T \mathbf{a}_i \tag{4.8}$$

Here $\mathbf{a}_i$ represents the vector containing the information about the potential key sample in feature space. This means that $\mathbf{a}_i$ is a column vector of $\mathbf{X}_{potential}$ with the index i for column indication. With this vector $\mathbf{r}_{new}$ it is possible to calculate a new column vector for the orthonormal base of $\mathbf{Q}$.

$$\mathbf{q}_{new} = \mathbf{a}_i - \mathbf{Q}\mathbf{r}_{new} \tag{4.9}$$

In Figure 4.2 these calculations are shown graphically.

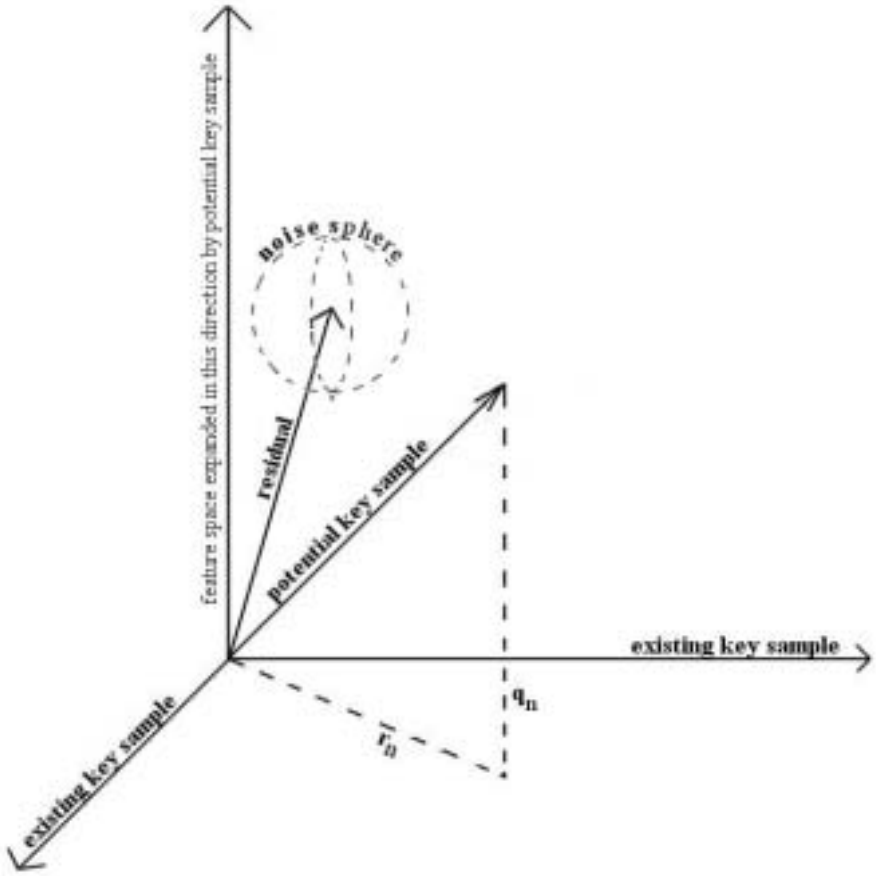

Figure 4.2. Graphical representation of key sample inclusion criterion (step 1)

The base plane is spanned by two existing key samples. The residual is a vector, which has a component in the direction that cannot be projected on the base plane. The feature space will be extended in this direction when the potential key sample is added to the set of existing key samples. The potential key sample first is projected onto the existing feature space (base plane), which results in vector $\mathbf{r}_{new}$. Vector $\mathbf{q}_{new}$ is calculated by subtracting $\mathbf{r}_{new}$ from the potential key sample vector. The selection criterion consists of two steps. First the length of $\mathbf{q}_{new}$ is calculated. When this length is too small the potential key sample is too similar to the existing key samples. Or in other words the potential key sample does not span the feature space in a significant new direction. When the length of $\mathbf{q}_{new}$ is large enough the second step in the criterion is tested. The incoming samples are corrupted with noise. Therefore the residual vector has an uncertainty region. In Figure 4.2 this uncertainty is drawn as a (noise) sphere. In reality this uncertainty region will probably not be a sphere. However to show the second step in the criterion it is useful to draw a similar figure, in which the vectors have less dimensions. This is done in Figure 4.3.

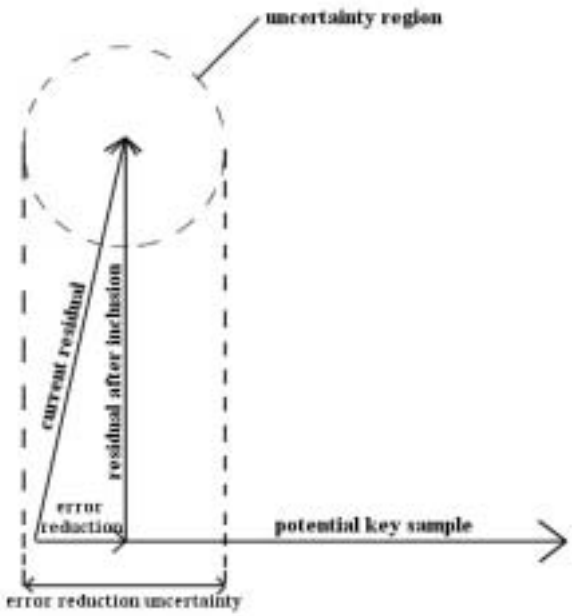

Figure 4.3. Graphical representation of key sample inclusion criterion (step 2)

Here the current residual, the potential key sample, the residual after the inclusion and the error reduction are drawn as vectors. There is an uncertainty to the current residual because of the noise in the output values of the samples. In Figure 4.3 this uncertainty is rather large compared to the error reduction. Therefore the error reduction uncertainty is large as well. The second step in the inclusion criterion compares the error reduction to its uncertainty. When the error reduction uncertainty is larger than the error reduction itself it is decided that noise is being fitted into the approximation, so the potential indicator will not be included. This step of the selection criterion is exactly the same as in the offline algorithm and is described in a more mathematical way in paragraph 3.1.5. The selection and inclusion of potential key samples is repeated until the remaining potential key samples do not meet the criterion anymore. Then the algorithm continues with the selection of key samples that possibly can be omitted.

4.5 Key Sample Omission

The selection and omission of key samples is done in exactly the same way as is explained in paragraph 3.2.7. First the key sample, which contributes least to the approximation, is selected. The column of matrix **R** corresponding to this key sample, is shifted to the right end side of the matrix. By means of a Givens rotation the matrix **R** is made upper-triangular. The last element of vector $\bar{\mathbf{z}}_{\mathbf{X}}$ in (3.69), is the weighted error increase due to the removal of the key sample. When the value of this element is larger than the specified omission criterion value the key sample will not be omitted.

5 Experiments

In this chapter the Blocked Key Sample Machine, described in chapter 4 will be tested in a simulation environment and on a real physical system. The simulations will be executed in 20-sim (Controllab products, 2005). The algorithm will be tested on the system depicted in Figure 5.1. This system is treated in detail in (Dirne. H., 2005).

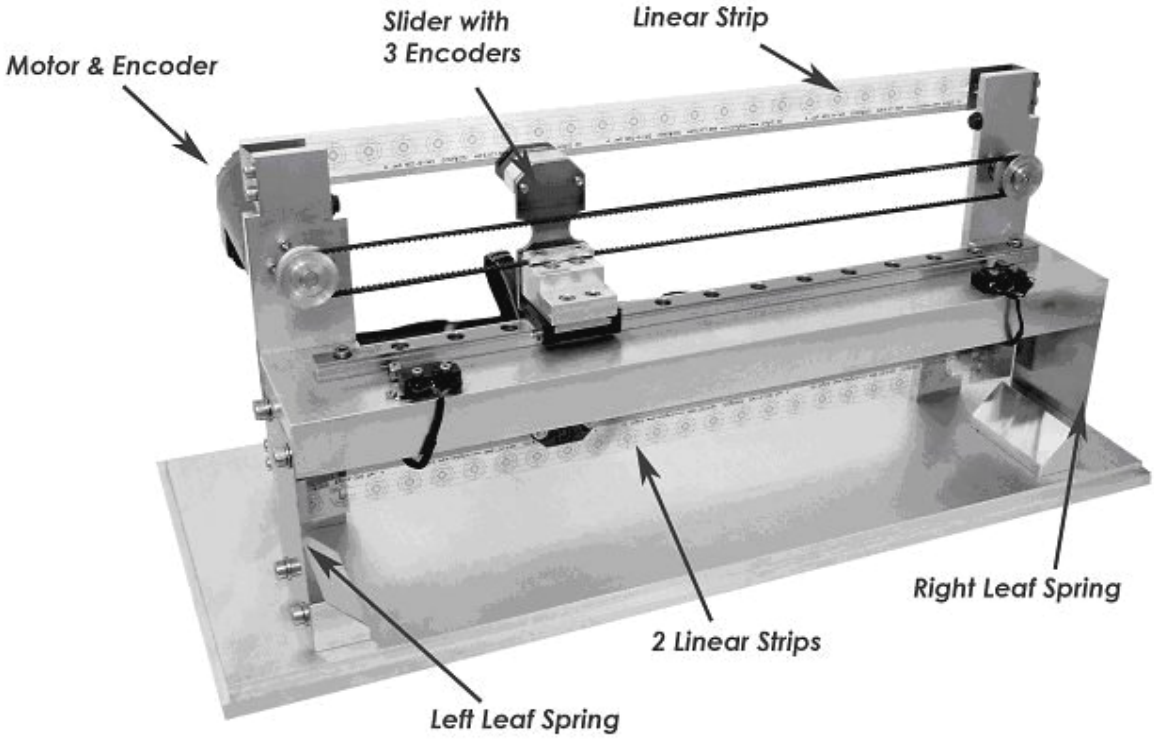

Figure 5.1. Physical System

The system consists of a motor that drives a belt on which a slider is mounted. The frame on which the slider moves is mounted on two leaf springs. Therefore the position of the slider with respect to the frame as well as the position relative to the fixed world is measured.

The BKSM algorithm will only be tested in a repetitive motion configuration. For random motion testing, a training set has to be generated in which the samples are well distributed over the input space. The input space consists of three variables, these are the first, second and third derivative of the reference signal. The reference signal and its derivatives correspond to the position, the velocity and the acceleration of the load. For obtaining a training set in which the samples are well enough distributed over the input space, a random path generator is needed. This random path generator collects samples of the target function at different positions with different velocities and accelerations. For repetitive motion it suffices to have a one-dimensional input space in which motion time is used as input value for the training samples. Due to time limitations, the BKSM algorithm is only tested for repetitive motion

5.1 Simulations

For simulation purposes, a model of the physical system must be derived. This model, which is taken from (Dirne. H., 2005), is depicted in Figure 5.2. The input signal is multiplied by two constants, i.e. the amplifier and motor constant. This results in a force generated by the motor. This force is exerted on the frame mass and the motor inertia. The frame is connected to the fixed world by a spring-damper, which represents the leaf springs in Figure 5.1. The load mass (slider) is

connected to the motor inertia by a spring-damper, which is in the physical system the rubber belt, connecting the motor to the slider. The position of the load mass with respect to the frame is used as output for the model.

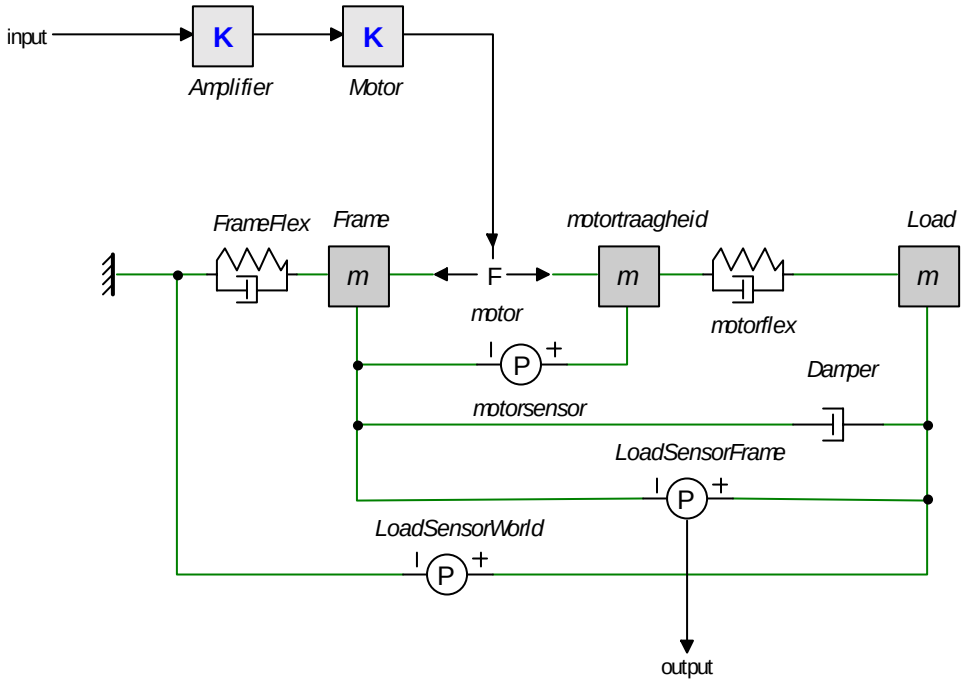

Figure 5.2. Model of the physical system in Figure 5.1

By using a control loop, the position of the load with respect to the frame is controlled. This is shown in Figure 5.3

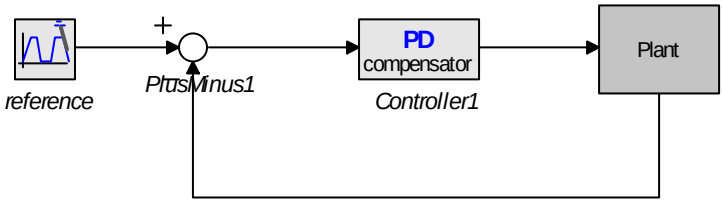

Figure 5.3. Control scheme

In the figure above the block “Plant” represents the model of Figure 5.2. The position of the load mass is the output of this plant. This position is subtracted from the reference signal. The result is the error between the reference position and the real position. This error is fed to the PD controller, which can steer the plant to the desired state. The output signal of the PD-controller indicates the amount of force the motor has to generate. However, the input of the plant is not directly connected to the motor force. The input signal is first multiplied by the amplifier and motor constant. Therefore, it is necessary to multiply the output signal of the PD-controller by the inverse of these constants, by which the transfer function between PD-controller output and motor force becomes equal to one. In that case the output signal of the PD-controller is equal to the force to be generated by the motor. The multiplication with the inverse of the amplifier and motor constants is executed inside the block “Plant”. The reason for using a PD-controller instead of a PID-controller or another controller is explained later on. The PD-controller parameters are calculated with the PID-controller configuration rules from (Vries, T.J.A. de, 2003). The PD-controller will be set in the same way as the PID-controller, except that the integrating action will be left out. The required crossover frequency is determined with:

$$\omega_c = \frac{1}{t_m}\sqrt[3]{\frac{32h_m}{\alpha\|e_{pos}\|}} \tag{5.1}$$

The crossover frequency calculated is based on a skew-sine reference signal. In (5.1) h_m is the maximum amplitude of the reference signal, which in this case is 0.07 [m]. Parameter t_m is the duration of the transient in the skew-sine. For the reference signal used in the experiments t_m has a value of 0.25 [s]. The period time for the reference signal is 2.0 [s]. The ratio between the values of differentiation tau τ_d and high-frequency roll off tau τ_h is represented by α. The value for α is in this simulation chosen to be equal to 10. The position error is chosen to have a maximum of 1.10^{-3} [m]. When these parameter values are used in (5.1) the required crossover frequency for the closed-loop system is 112.76 [rad/s]. The model of the physical system is simplified as a "moving mass model" (Vries, T.J.A. de, 2003), with a mass m_{load} of 0.3 [kg]. The PD-controller settings are calculated with:

$$\tau_d = \frac{\sqrt{\frac{1}{\alpha}}}{\omega_c} = 0.028 \tag{5.2a}$$

$$\tau_h = \frac{\sqrt{\alpha}}{\omega_c} = 0.0028 \tag{5.2b}$$

$$k_p = \frac{m_{load}\omega_c^2}{\sqrt{\frac{1}{\alpha}}} = 1.2062 \cdot 10^3 \tag{5.2c}$$

The simulation results of the control scheme of Figure 5.3, with the PD-controller settings calculated above are shown in Figure 5.4. In this figure the reference signal, the position error, the PD-controller output and the motor amplifier output are shown. The position error in the simulation has a maximum of $2.54\ 10^{-3}$ [m]. However, the maximum tracking error was chosen to be $1.00\ 10^{-3}$ [m]. This difference is due to the fact that the PD-controller settings are calculated for a simplified model, i.e. a moving mass system. The model of the physical system is more complicated; this results in a difference between the maximum tracking error and the chosen maximum value for this error. The motor allows a maximum input current of about 2.75 [A]. The maximum motor current used in this control system is 0.64 [A].

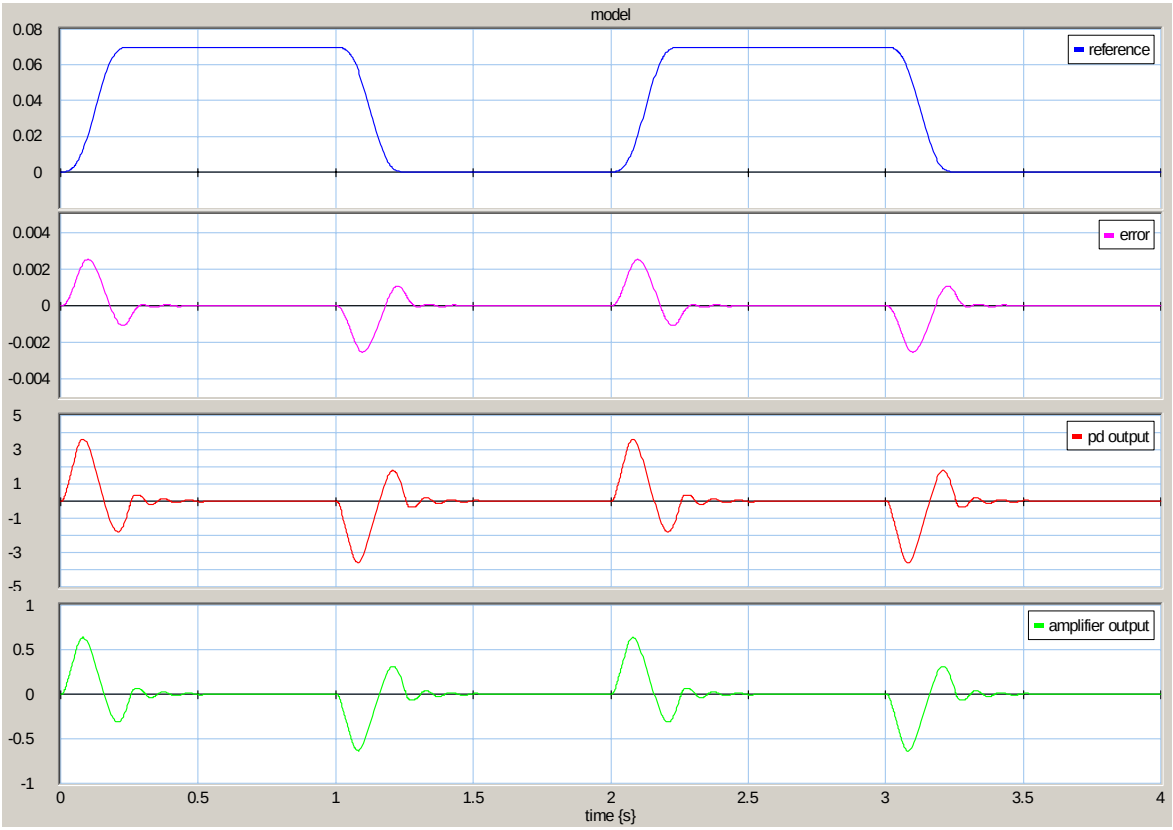

Figure 5.4. PD-controller simulations results

The control system can be extended with a feed forward controller. As in the PD-controller the plant is considered to be a moving mass system. The feed forward controller can be implemented by multiplying the second derivative of the reference signal (the disired acceleration) with a constant. This constant should have the value of the total moving mass. The output of the feed forward controller is added to the output of the PD-controller. This control scheme is depicted in Figure 5.5.

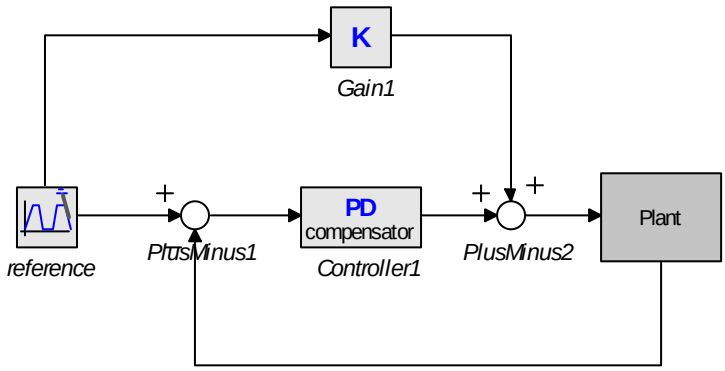

Figure 5.5. Extended control scheme

The feed forward controller consists of *Gain1*, with the second derivative of the reference signal as input. The value of *Gain1* is 0.3 because the load mass is 0.3 [kg]. The simulation results are shown in Figure 5.6.

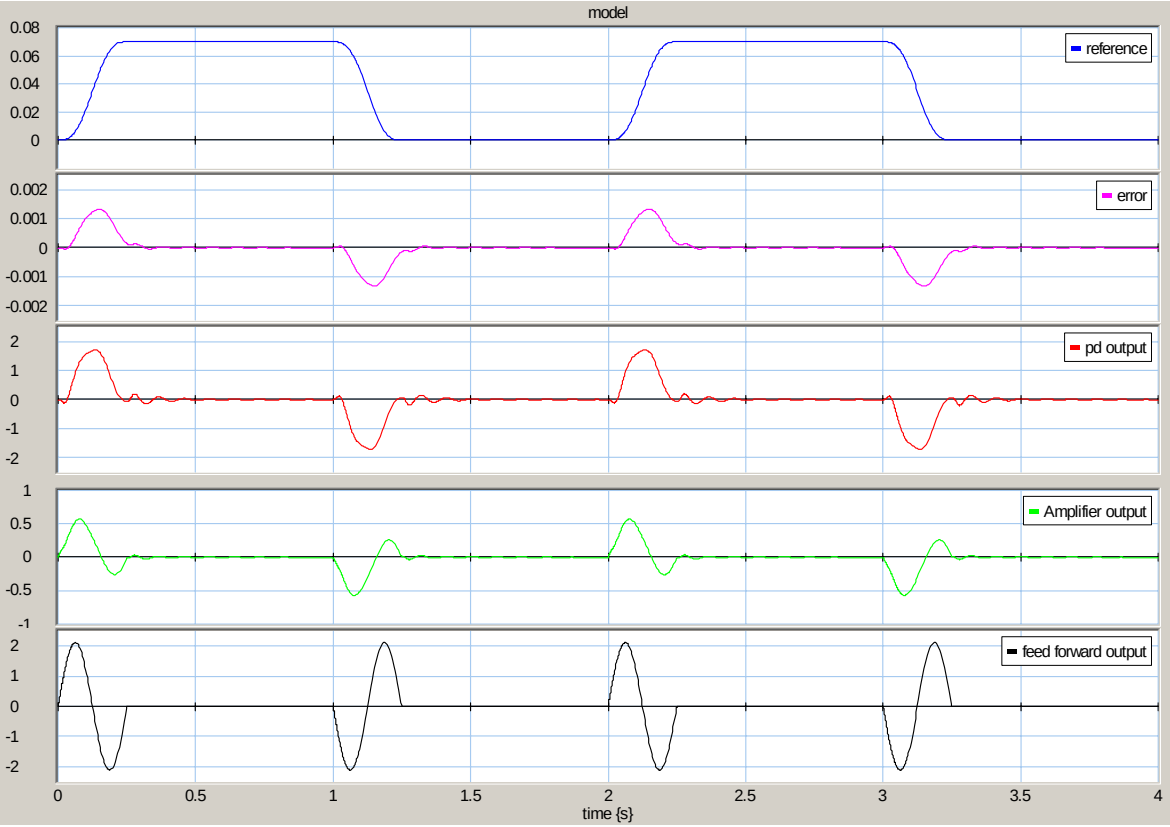

Figure 5.6. Simulation results from control scheme Figure 5.5.

The figure above shows from top to bottom the reference signal, the position error, the PD-controller output, the motor amplifier output and the feed forward controller output. These simulation results show a maximum tracking error of 1.3 10^{-3} [m]. The maximum tracking error is reduced by 50 % due to the feed forward controller. The maximum desired motor current is 0.57 [A], which is 10 % smaller than the maximum motor current in the control scheme without a feed forward controller.

In the following experiment the feed forward controller used in the control scheme of Figure 5.5 is replaced by a BKSM feed forward controller. The position error could be used as target function for the BKSM. When the error is known in advance, the system can be controlled in a way that circumvents the occurrence of the error. Although complete error elimination is in practice not possible due to noise and non-repetitive disturbances, the error can be reduced significantly. However, the error signal does not contain information about the system being controlled. Therefore it is necessary to implement a system-based filter, which scales the different frequency components of the error signal. Instead of constructing such filter from scratch, it is quicker using one that is already present, namely the PD-controller. In Figure 5.4 the PD-controller output is very similar compared to the position error. The error signal is scaled so it can be used on this particular moving mass system. The derivative action in the PD-controller amplifies the higher frequencies more compared with the lower frequencies. This derivative action results in a phase-lead of the controller output with respect to the error signal. This phase-lead implies that the controller is somewhat ahead of the actual occurring position error. Therefore, the controller can somewhat compensate for an error before it actually occurs. Because of the scaling and the phase-lead the PD-controller output is more useful to be used as a target function for the BKSM than the error signal. The phase-lead, is also the reason for using a PD-controller instead of a PID-controller. The integral action in this controller results in a phase-lag, which cancels out the phase-lead of the derivative action. In this experiment the output of the PD-controller shown in Figure 5.4 is used as the target function. Therefore, time spans the input space and the PD-controller output is the output space of the target function. Because of to project time constraints the “learning” of the function is done by using the existing code in Matlab.

The samples are saved into a text file and loaded by Matlab. The Matlab code determines which samples become key samples and what the corresponding weight has to be. The PD-controller output signal from Figure 5.4 has been approximated by the BKSM. This is shown in Figure 5.7.

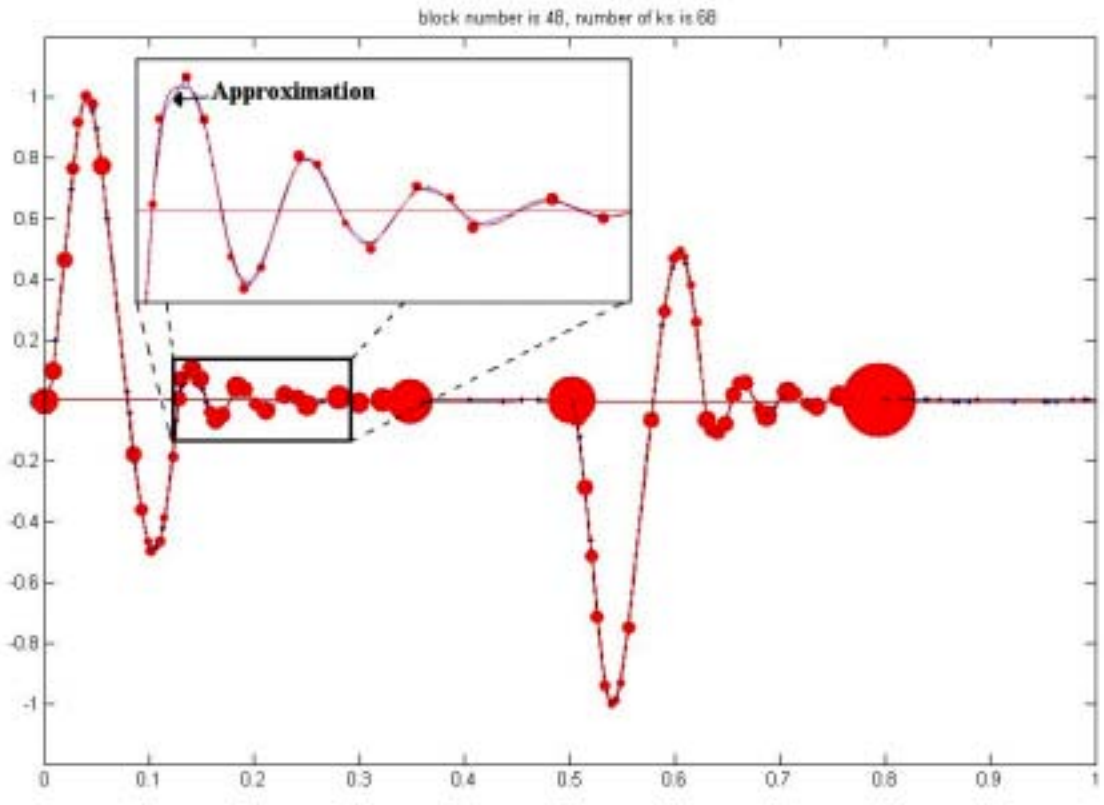

Figure 5.7. Approximation of PD-controller output by BKSM

In the approximation shown above the block size of the BKSM was set to 100 samples. The total training set consisted of 4800 samples. The BKSM uses 68 key samples for the approximation of the function, which are depicted as dots. The dot size shows how many samples a key sample represents. The approximation and the function are so similar that differences can only be seen in the zoomed part of Figure 5.7. Furthermore the figure shows that the function is scaled to have an input space of [0,1] and an output space of [-1,1]. The reason for this is that when having multiple inputs, all inputs are equally important for the approximation. Although the target function in Figure 5.7 is scaled, this particular experiment could well be executed without the scaling of the target function, because the input space is one-dimensional. The key samples, weights and the scaling factors are stored in text files, which are loaded into 20-sim. Time is used as an input for the prediction of the function value. The prediction is for simulation duration reasons done in 20-sim. The output of the BKSM is added to the output of the PD-controller. This control setup is shown in Figure 5.8.

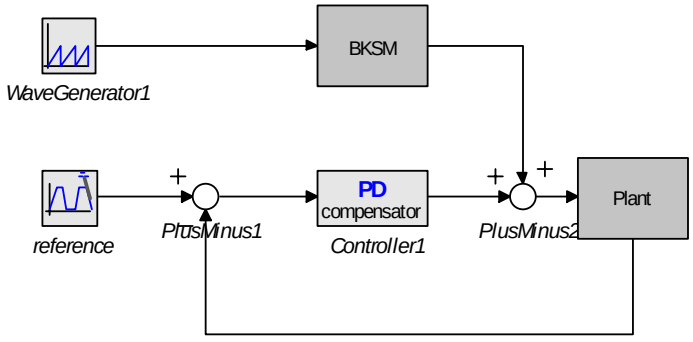

Figure 5.8. Control scheme with an BKSM and a PD-controller

The *WaveGenerator1* in the figure above is used to generate time samples for the input of the BKSM. The reason time is used as an input, instead of the reference signal is that the latter is not bijective. Therefore one reference input value could have two or more prediction output values, this makes the reference signal not the best candidate for being used as an input. Figure 5.9 shows the simulation results of the control scheme of Figure 5.8.

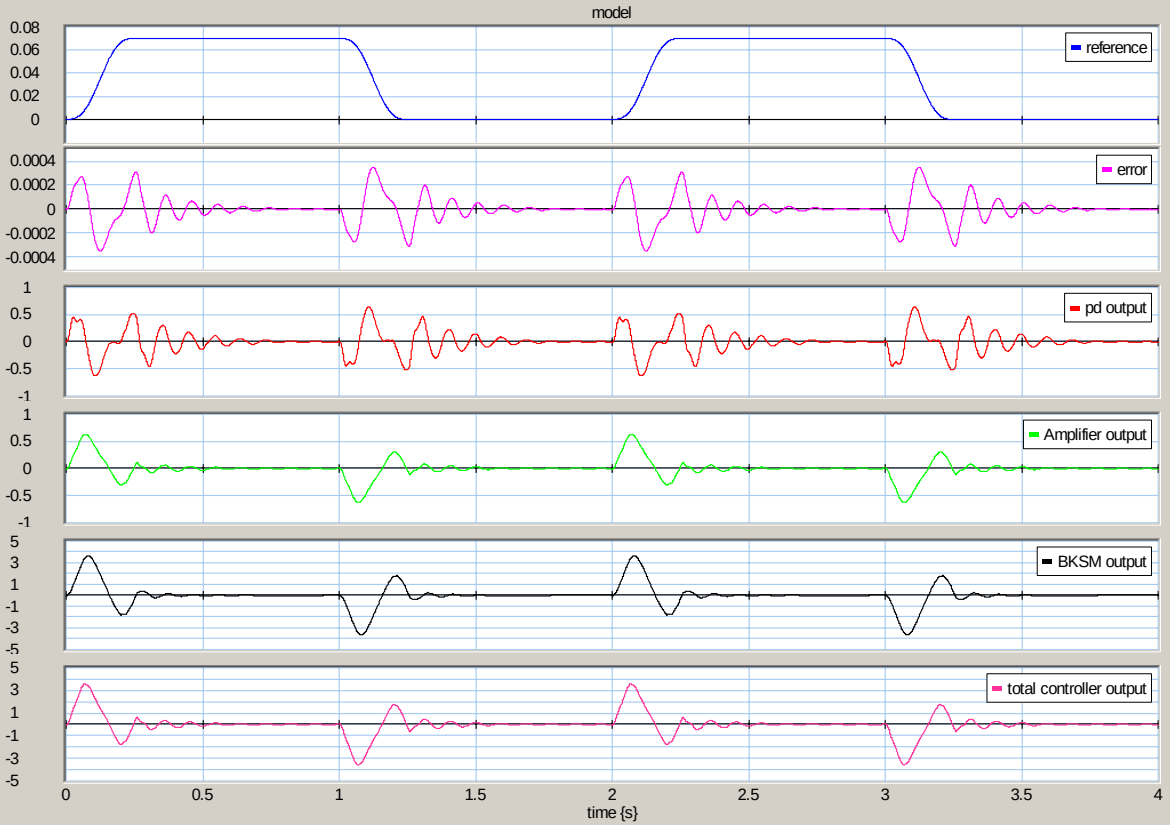

Figure 5.9. Simulation results from control scheme Figure 5.8

The figure above shows from top to bottom the reference signal, the position error, the PD-controller output signal, the motor amplifier current, the BKSM output signal and the total controller output signal. These simulation results show a maximal position error of 3.5 10^{-4} [m]. Compared to the maximal position error of the PD-controller control scheme this is an error reduction of more than 80 % . The maximal motor amplifier current is 0.62 [A]. The results also show that the PD-controller still has a significant output. The next step is to iterate the learning process of the BKSM until noise is being fitted into the approximation and no further improvement can be made.

In the second iteration the total controller output, which consists of the PD-controller output added to the BKSM output, is considered to be the new target function. The target function has been approximated with 96 key samples. The maximum amount of key samples is set to 100 for calculation speed reasons. In Figure 5.10 the BKSM has approximated the total control output from Figure 5.9.

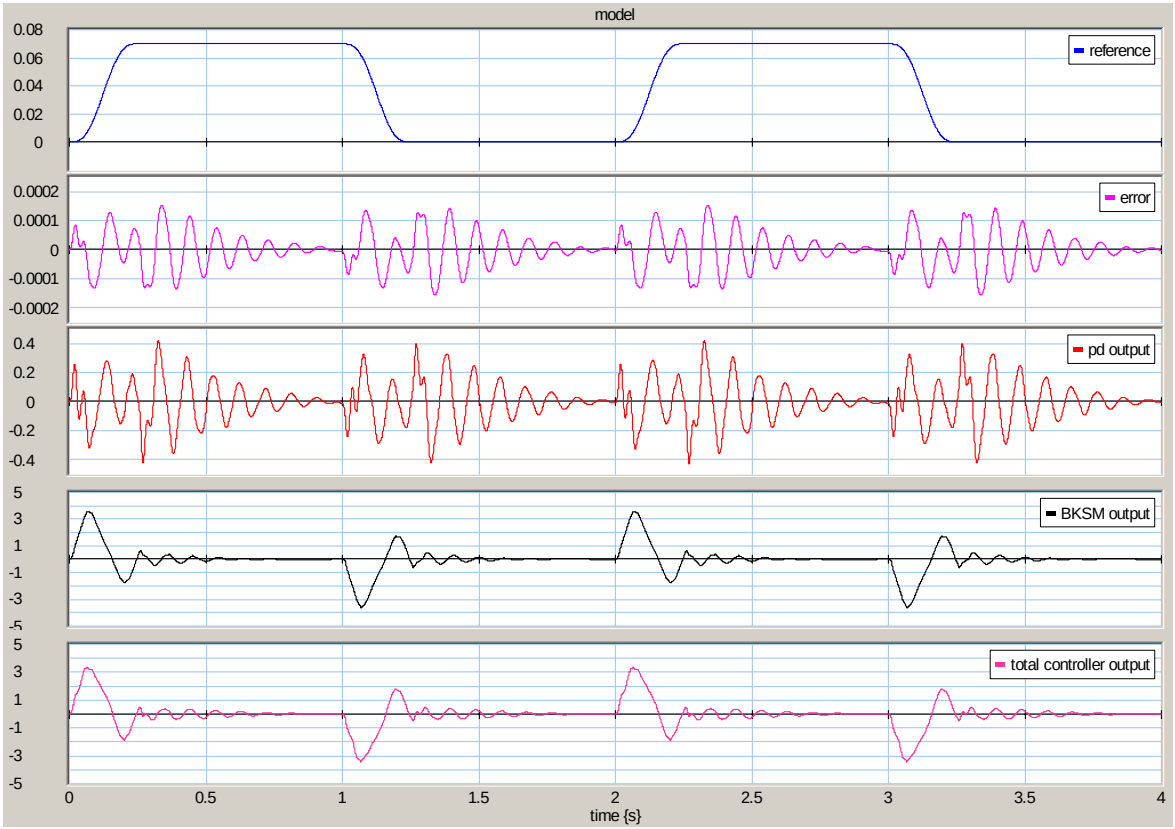

Figure 5.10. Simulation results of second BKSM iteration

The motor amplifier output is not shown in the figure above for surveyability reasons. The maximum position error is in this simulation equal to 1.5 10^{-4} [m]. This means that the position error has been reduced in the second iteration by 40 % compared to the first iteration.

The iteration is then executed. The total control signal of Figure 5.10 is approximated by the BKSM. The result is shown in Figure 5.11. The number of key samples is restricted to 100 for limited memory and calculation speed reasons.

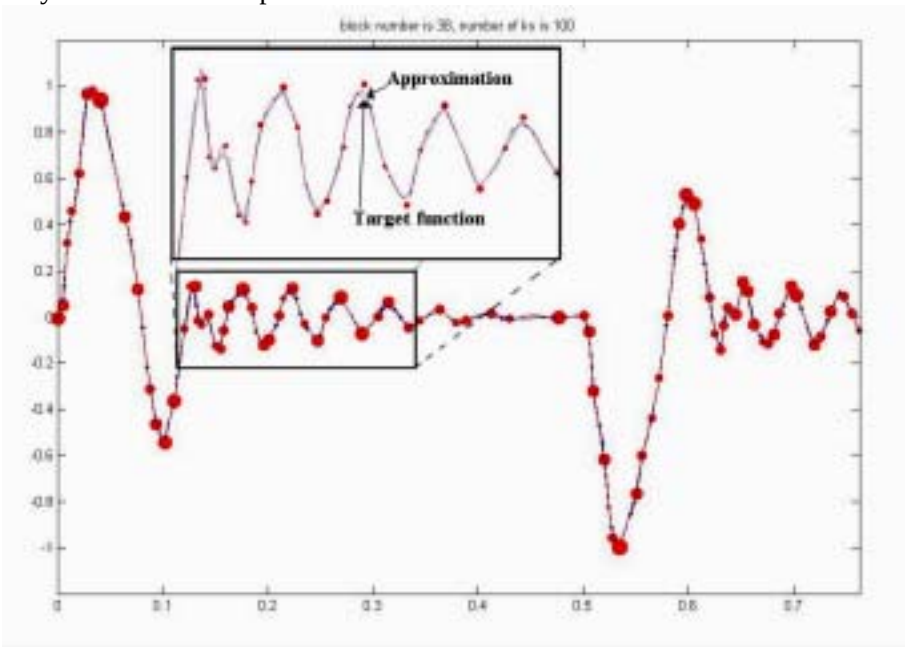

Figure 5.11. Approximation of the 3e BKSM iteration

When this approximation is used in the BKSM controller the results of Figure 5.12 are obtained.

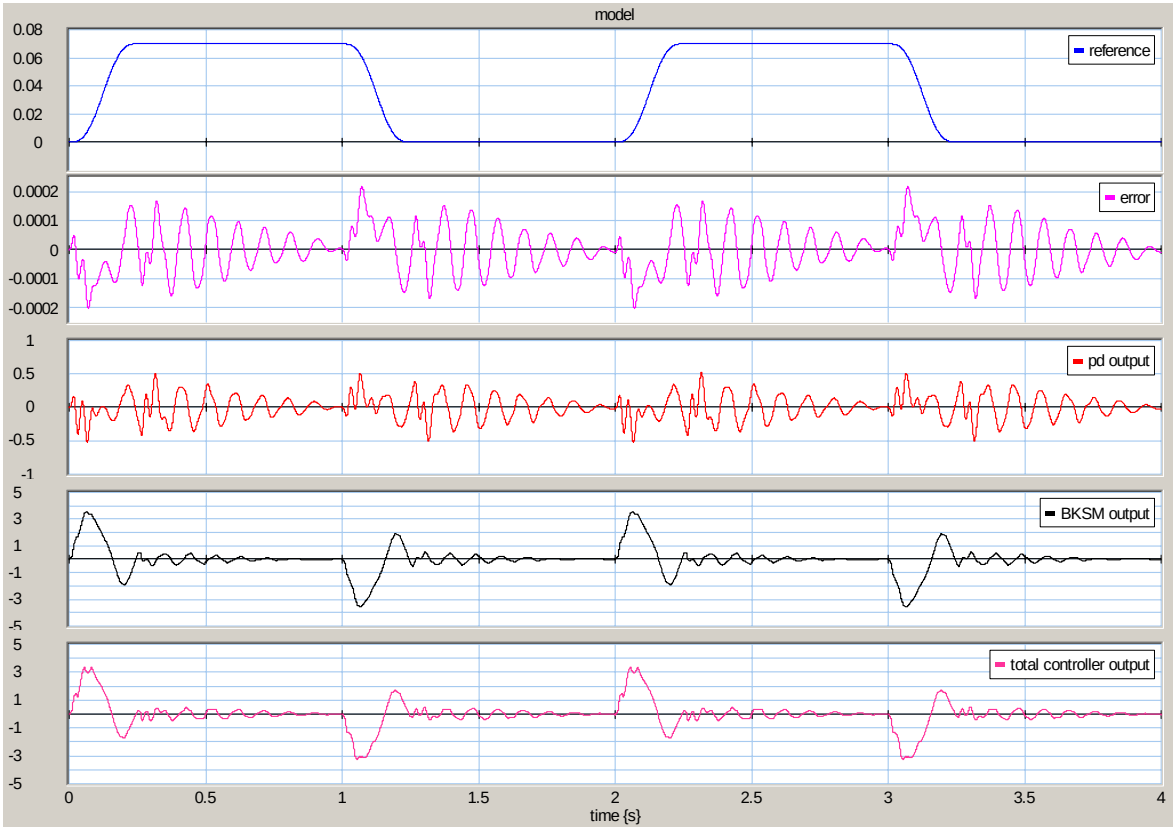

Figure 5.12. Simulation results of third BSKM iteration

Although the approximation seems to be accurate as can be seen in the zoomed part of Figure 5.11, the maximum of the position error has increased compared to the previous simulation. The maximum position error has a value of 2.2 10^{-4} [m]. The information, which is incorporated into the approximation between the second and third iteration, has a negative effect on the performance and therefore it can be regarded as noise.

The experiment above is repeated for block sizes of 10 and 250 samples per block. The results are shown in Table 5.1.

Iteration step number	Maximum Position Error in meter		
	10 sample block size	100 sample block size	250 sample block size
1st	3.4 10^{-4}	3.5 10^{-4}	3.2 10^{-4}
2nd	1.6 10^{-4}	1.5 10^{-4}	1.5 10^{-4}
3rd	2.0 10^{-4}	2.2 10^{-4}	1.5 10^{-4}
4th	Not simulated	Not simulated	1.4 10^{-4}
5th	Not simulated	Not simulated	1.7 10^{-4}

Table 5.1. Overview of simulation results with different block sizes

These results show that the maximum position error decreases initially with each iteration. When the iteration is continued, noise is being fitted and the maximum position error will increase. Simulations are executed until the 4th iteration step for block sizes of 10 samples and 100 samples. Continuing the iteration would only increase the position error further. The simulations with the block size of 250 samples shows that the BKSM performs best with a relative large block size and that it is less sensitive for incorporating noise into the approximation. The reason for this is probably that the larger the block size, the better the samples can be compared to each other. Therefore samples containing little information about the function have less influence on the approximation. The best result is after four iteration steps with a block size of 250 samples. The reduction of maximum position error of the BKSM is compared to the regular PD-controller over 90 %. The simulation results of the best performance are shown in Figure 5.13.

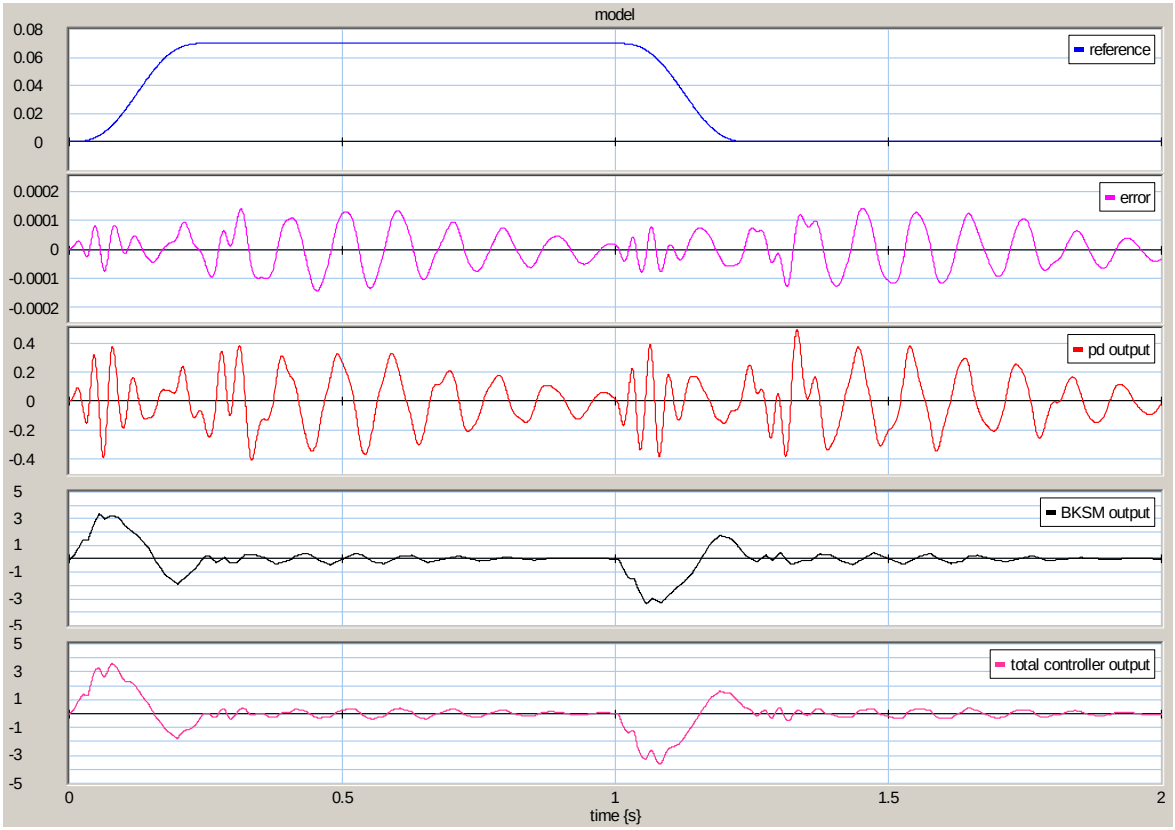

Figure 5.13. Simulation results of 4th iteration of BKSM with a block size of 250 samples

Although these results are obtained by simulation, it is very likely that there will be a significant error reduction when tested on real physical system.

5.2 Testing BKSM on a physical system

As in the simulations, the testing of BKSM on a physical system is done for a repetitive motion. For testing the BKSM algorithm on the physical system shown in Figure 5.1 the control scheme of Figure 5.3 has to be changed. The plant, which contains the model of the physical system, is replaced by the physical system itself. Outputs to and encoders from the PD-controller translate the input and output from the physical system. Furthermore, the analogue PD-controller used in the simulations of 5.1, is replaced by a discrete PD-controller with a sample frequency of 1 [kHz]. The control scheme used for testing on the physical system is shown in Figure 5.14.

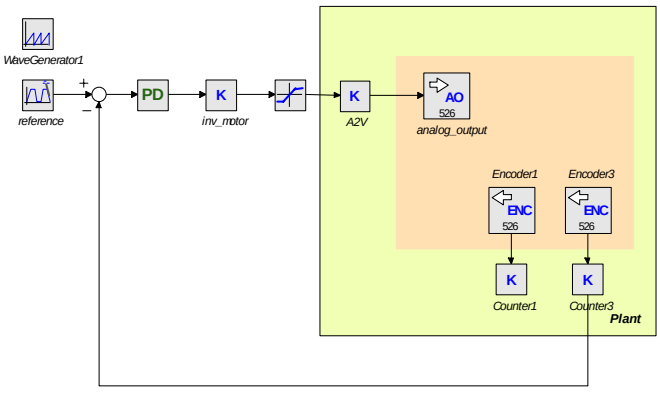

Figure 5.14. Control scheme for testing BKSM on the physical system of Figure 5.1.

The discrete PD-controller is used in the same configuration settings as done with the analogue controller in the simulations. These controller settings can be used in the discrete controller, because the sample frequency is relative high compared to bandwidth of the physical system. The wave generator is used for generating a time index, by which the logged data can by synchronized. The encoder, which is used as the system output, encodes the load position with respect to the frame. In Figure 5.15 the results of this control scheme are shown.

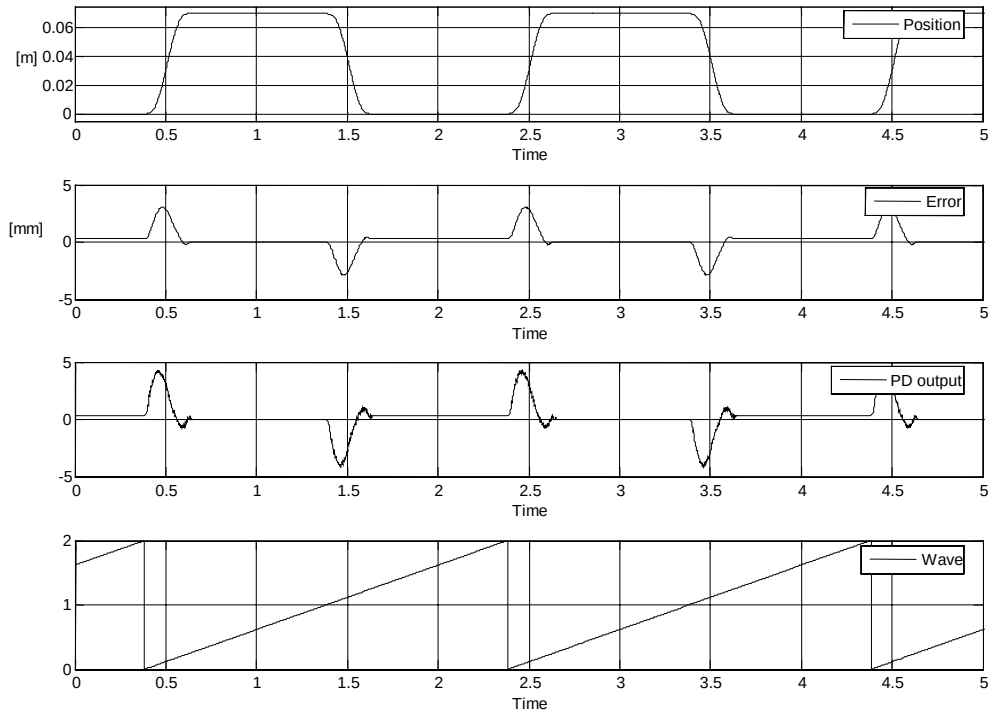

Figure 5.15. Test results of the control scheme of Figure 5.14

In this figure the reference signal, the position error, the PD-controller and the wave generator output are depicted. The maximum potion error is 3.100 10^{-3} [m]. The position error has a static component, which is not present in the simulation results. The reason for this is that the coulomb friction and stiction are not incorporated into the model used in the simulations. The size of the maximum static position error is 3.000 10^{-4} [m].

In the following experiment the discrete PD-controller is extended with the BKSM feed forward controller as is done in the simulations. The control scheme with this extended controller is shown in Figure 5.16.

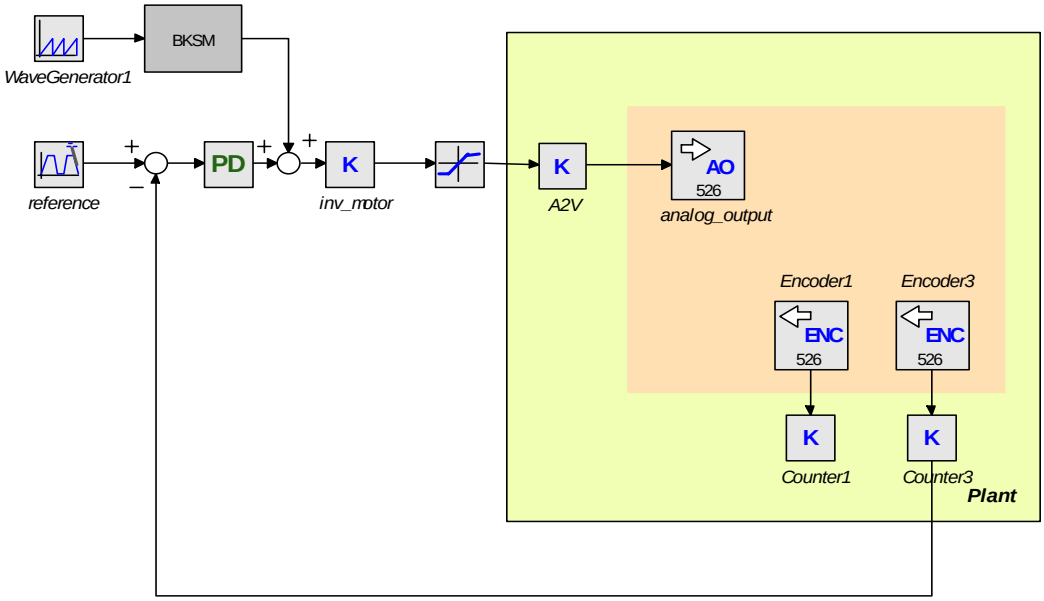

Figure 5.16. Control scheme with extended controller

The BKSM has been trained to approximate for the PD-controller output of Figure 5.15. The block size used by the BKSM is set to 250 samples, because this gave the best results in the simulations. This approximation is depicted in Figure 5.17.

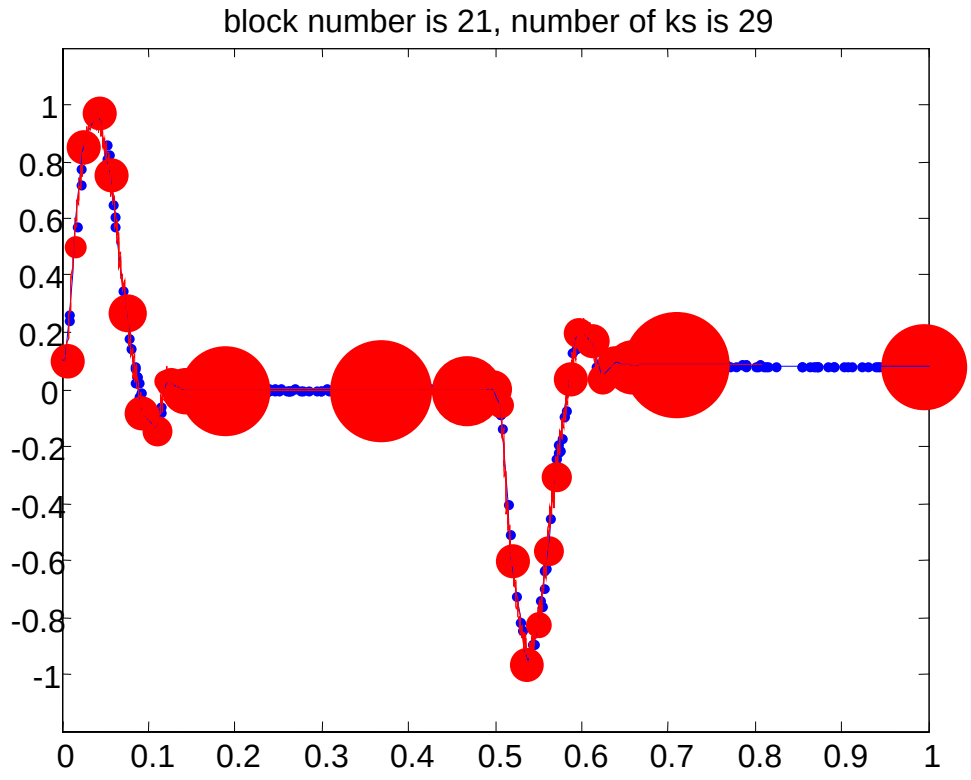

Figure 5.17. BKSM approximation of PD-controller output of Figure 5.15

Key samples are depicted as dots in the figure above. The size of these dots is related to the amount of samples a key sample represents. This approximation is used in the BKSM of Figure 5.16, which gave the following results.

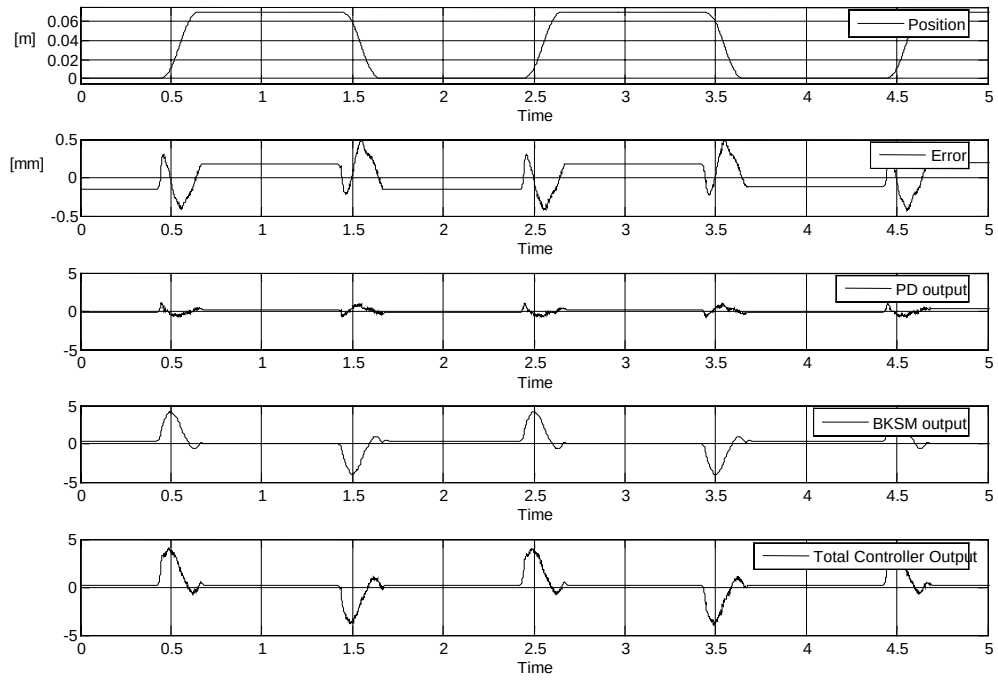

Figure 5.18. Test results of extended control scheme

These test results show a maximum position error of 5.1503 10^{-4} [m]. This is a decrease in maximum position error of 85 % compared to the PD-controller. The maximum static error is 1.75 10^{-4} [m]. That means that the static error is decreased with 40 % compared to the static error in the PD-controlled system.

As done in the simulations the learning of the BKSM is done in several steps. Therefore the total controller output signal is logged and used to as new target function for the BKSM. The test results of the second and third iteration steps experiment are respectively shown in Figure 5.19 and Figure 5.20.

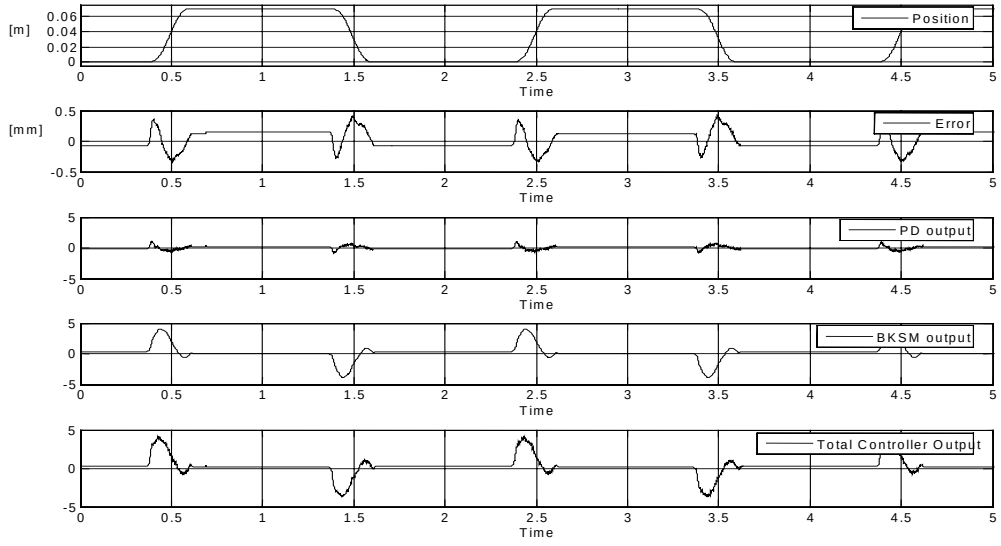

Figure 5.19. Test results of second BKSM iteration

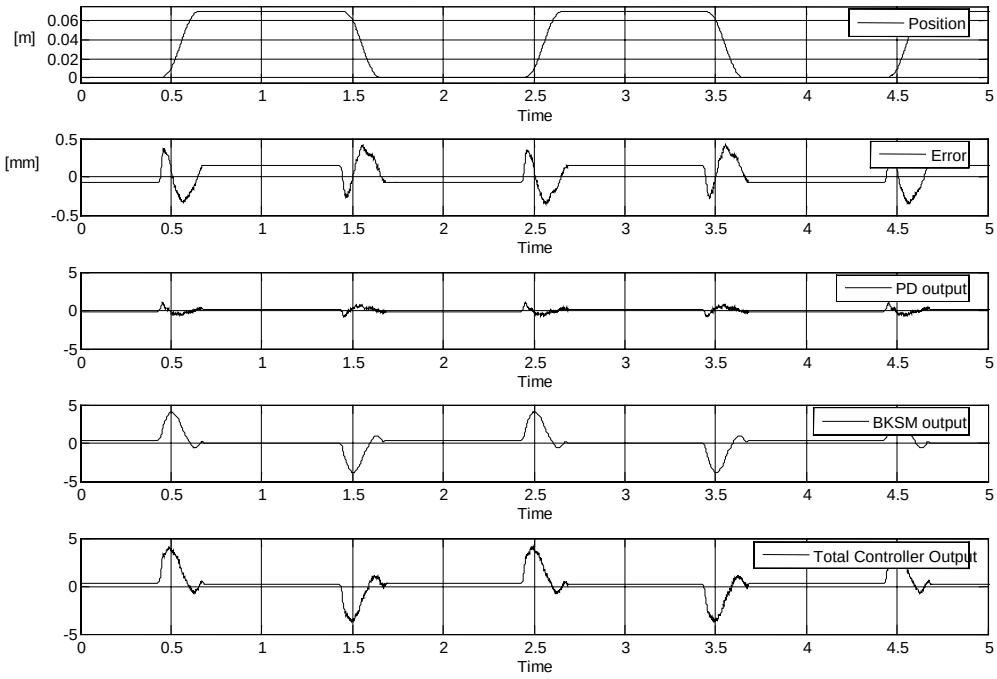

Figure 5.20. Test results of third BKSM iteration

Both test results show a maximum position error of 4.3024 10^{-4} [m]. The position error is not further decreased between the second and third iteration. Therefore no following iterations have been executed. The maximum position error is decreased with 86.1 % compared with the PD-controlled system. The maximum static position error is in both test results 1.5 10^{-4} [m]. This means that the maximum static position error is reduced by half.

The experiments described above are also executed with a block size of 100 samples per block. Two iteration steps have been made which both resulted in a maximum position error of 5.3024 10^{-4} [m] and a maximum static position error of 1.75 10^{-4} [m]. The second iteration did not result in a decrease of the position errors. Therefore no third iteration has been executed. The reason that few learning step suffices for obtaining the best performance is that the samples are corrupted with very small measurement noise. As shown in the simulation results a block size of 250 samples per block gives better performances than one of 100 samples per block. The most probable reason for this is that the algorithm can evaluate samples better when they are compared to larger group of other samples. Therefore a BKSM with a larger block size is less sensitive for measurement noise and will approximate the target function more accurately.

6 Conclusion and Recommendations

This project had five project goals, which are described in the introduction. In this chapter each goal and whether it was attained in this project will be discussed.

6.1 Literature study of existing function approximators

The literature study was mainly aimed at least squares and KSM approximators. Support vector machines have not been studied extensively, because it became clear in an early stage that studying these approximators would not have a large contribution in understanding the principals of KSM. Studying only the literature of the least squares and KSM approximators did not suffice in completely understand all details of these algorithms. Therefore two variants of least squares approximators as well as two variants of key sample machines were rebuilt in Matlab. Although the rebuilding process was very time consuming, it resulted in a detailed insight of both algorithms. As described in literature it was found that the least squares algorithms were unable to approximate certain functions. The offline and online KSM algorithms are much more complex than the least squares algorithms but can however approximate these functions. Although KSM approximators have a complex structure, they use computer resources in an efficient way. Assessing how well KSM performs compared to other approximators is quite difficult. The approximators have to be tested at least on approximation speed, accuracy, and efficiency. The performance of an approximator depends however on the target function, the used kernel function(s) and for the new developed KSM the number of samples per block (see paragraph 6.2). This makes the benchmarking of function approximators difficult undertaking. However, for any useful comparison between approximators benchmarking has to be done.

6.2 Designing and building a new KSM variant in Matlab

The requirements for the new KSM algorithm were that it should approximate recursively and incorporate the information of a variable amount of samples into the current approximation. The new KSM algorithm (BKSM) handles a block of samples in a way similar to the offline algorithm, but can update an existing approximation as is done in the online variant. When the principals of the online and offline KSM are understood, the BKSM algorithm will not hard to comprehend. The BKSM algorithm gave good approximation results and it is capable of handling samples much quicker than the online KSM. The block size used in the BKSM influences its performance. A small block size (about 10 samples per block) results in more computations and therefore the approximation becomes slower than when the block size is set to 250 samples. Further research has to be done in finding the optimal block size for having the quickest approximation.

6.3 Programming the BKSM algorithm in C or C++

This project goal was set for testing purposes. Matlab is ideal for developing algorithms but calculations are done much more swiftly when the algorithm is programmed in a C or C++ environment. The Matlab code has been automatically translated into C++ and compiled into an executable file. Although this file worked properly, it is not used in the experiments. The reason for this is that implementing the BKSM into the controller of the system in Figure 5.1, would require extending the system with a parallel computer on which the BKSM computations are done. Another reason for programming the BKSM in C++ is that 20-sim (Controllab products b.v.) can be extended with a KSM option. Because of time limitations this is not done within this project. The priority has been given to simulating and experimenting with the BKSM algorithm. It was not

necessary to translate the Matlab code into C or C++. The learning is done offline in Matlab and the predictions are calculated real time in 20-sim.

6.4 Testing the new key sample machine by means of simulation

The BKSM algorithm was only tested for repetitive motion. For testing with random motion it is necessary to obtain training sets with well-distributed samples, which contain information about the position, velocity and the acceleration dictated by the reference signal. Because obtaining these training sets is time consuming, the testing is limited to repetitive motion. Testing the BKSM was done with block sizes of 10, 100 and 250 samples per block. Although BKSM resulted in major performance improvements with all tree block sizes, the largest block size gave the best results. The maximum position error with the PD-controlled system was $2.54\ 10^{-3}$ [m]. When the controller was extended with the BKSM feed forward controller the maximum position error decreased to a value of $1.4\ 10^{-4}$ [m] after four iteration steps. This is means that the maximum position error is decreased with 95 %. During the simulations no static position error is encountered because the coulomb friction and the stiction are not modeled in the plant. The first BKSM iteration steps result in a decrease of position error, but after a certain amount of iterations the position error start to increase. The cause for this is that measurement noise from the samples is fitted into the approximation. The smaller the used block size, the more noise sensitive the BKSM will be. Therefore it is necessary that extensive research will be done to obtain some sort of procedure, which determines the block size and the number of iterations, which result in the best performance.

6.5 Testing of the new key sample machine on a physical system

The simulation results were promising for some good test results. The testing has been done for a block size of 100 and 250 samples per block. As in the simulations the largest block size resulted in the best performance. The maximum position error by the PD-controlled system was $3.100\ 10^{-3}$ [m] with a maximum static error of $3.00\ 10^{-4}$ [m]. When the controller was extended with the BKSM feed forward controller with a block size of 250 samples, the maximum position error was $4.3024\ 10^{-4}$ [m] and a maximum static position error of $1.50\ 10^{-4}$ [m], after two iterations. This means that the maximum position error is decreased by 85 % and the static position error by 50 %. Because the samples are contaminated with small measurement noise, it only takes two iterations for obtaining the best results.

6.6 General Conclusions

Besides the five project goals that are treated above, this project had another unofficial goal. This goal was to make the whole theory about key sample machines more accessible for following students. The theory was already described in the PhD. thesis of De Kruif (De Kruif, 2004), but there was a gap between the described theory and how the KSM algorithm was implemented in practice. In this M.Sc. thesis all steps of the KSM implementation are described in detail. Furthermore, the Matlab codes of the three KSM algorithms have been incorporated into the appendix chapter. Each code line has been commented and references to the text have been put in. Reading the theory described in this thesis in combination with the code, will certainly speed up the process of learning the KSM principals.

7 Appendices

Appendix A – Givens Rotation

The Givens rotation (Golub, and Van Loan, 1996) calculates c=*cos(θ)* and *s=sin(θ)* when given scalars *a* and *b*, such that:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \tag{A.76}$$

It is not necessary to calculate the angle θ for calculating *s* and *c*, so the rotation algorithm does not involve inverse trigonometric functions. The values of *s* and *c* are calculated with the following formulas:

$$b = 0 \quad \Rightarrow \quad c = 1 \qquad s = 0 \tag{A.77a}$$

$$|b| > |a| \quad \Rightarrow \quad s = \frac{1}{\sqrt{1+\left(\frac{a}{b}\right)^2}} \quad c = -\frac{sa}{b} \tag{A.77b}$$

$$|b| \le |a| \quad \Rightarrow \quad c = \frac{1}{\sqrt{1+\left(\frac{b}{a}\right)^2}} \quad s = -\frac{cb}{a} \tag{A.77c}$$

This procedure can also be applied on matrices. For example, a Givens rotation is applied on matrix **A** [m x n], in which X denotes a non-zero element:

$$\mathbf{A} = \begin{bmatrix} X & \cdots & X & \cdots & X & \cdots & X \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & X & \cdots & X & \cdots & X \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & X & \cdots & X & \cdots & X \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & X \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \end{matrix} \tag{A.78}$$

The row *k* contains non-zero elements, which prevent the matrix from being upper-triangular. Then a matrix **G**(I,j,θ) [m x m], can be constructed:

$$\mathbf{G}(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix} \qquad \text{(A.79)}$$

$$\begin{matrix} & & i & & k & & \end{matrix}$$

By multiplying matrix **G** with matrix **A**, the rows *I* and *k* are effected. So after this multiplication matrix **A** will be such that:

$$\mathbf{A} = \begin{bmatrix} X & \cdots & X & \cdots & X & \cdots & X \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \overline{X} & \cdots & \overline{X} & \cdots & \overline{X} \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \overline{0} & \cdots & \overline{X} & \cdots & \overline{X} \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & X \end{bmatrix} \begin{matrix} \\ \\ i \\ \\ k \\ \\ \\ \end{matrix} \qquad \text{(A.80)}$$

The elements, which have been altered by the Givens rotation, are denoted with a bar.

Appendix B – Offline KSM Matlab code

Main Code

Creating the trainingset

```
clear all;                          % Clear memory

                                    % function initialization
n=250;                              % The training set will consist of 250 samples
sigma = sqrt(0.003);                % The sigma value of the Gaussian noise

v = 0.05;                           % Parameter used in the function
x = linspace(0,1,n)';               % Creating an array x with n equally space numbers
                                    % between 0 and 1
sx = size(x);                       % Determining the number of samples in array x

y = sin(1./(x+v))+sigma*randn(n,1); % Using the target function (3.3) for generating output
                                    % values for the samples. These are stored in array y
```

Algorithm initialization

```
'Function has been initialized, X Potential is being generated'

for i = 1 : sx(1)
   for k = 1 : sx(1)
      Xpot(i,k) = kernel(x(k), x(i));   % generating X_potential described in formula (3.7)
   end
```

```
end

sXpot = size(Xpot);                    % Determining the size of X_potential.

'X potential has been created'

X = [ones(sx(1),1) y];                 % Incicator matrix is filled with a column containing only
                                       % ones and is augmented with array y (as described in the
                                       % theory). The column with ones results in an
                                       % approximation where the offset of the function is
                                       % determined. This is however not necessary.

sX = size(X);                          % determining the size of indicator matrix X
R = chol(X'*X);                        % Matrix R of the QR-decomposition of X is equal
                                       % to the cholesky decomposition of (X^TX) see
                                       % formula (3.24)

b = R(1:end-1,1:end-1)\R(1:end-1,end); % The weight vector b is calculated by (3.19). The vector
                                       % z_x is here represented by R(1:end-1,end)

SSRegressionPrevious = n*mean(y).^2;   % The first residual is the mean of the output values
                                       % squared
'Adding keysamples'
```

Starting the actual approximation

```
while sX(2) < sXpot(2)                 % This while-loop is repeated until the number of
                                       % columns of X is equal to that of X_potential. When this is
                                       % the case, it means that all samples have become
                                       % key samples

   residu = y - X(:,1:end-1)*b;        % calculating the residual see formula (3.9)

   angle=(Xpot'*residu)./sqrt(diag(Xpot'*Xpot)); % angle between the residual and the potential
                                                 % key samples. See formula (3.10). As seen in
                                                 % this formula not the actual angle is calculated
                                                 % but the cosine of this angle is calculated

   [max_dummy,maxi] = max(abs(angle));           % determining the minimum angle. This is of course
                                                 % the maximum value for the cosine

   [X,R] = aug(X,R,Xpot(:,maxi));      % The potential key sample with the minimum angle is
                                       % included into the matrices X and R with the function
                                       % "aug". This function is shown below.

   b = R(1:end-1,1:end-1)\R(1:end-1,end);   % weight vector b is updated after the inclusion of the
                                            % the new key sample

   SSRegression = b'*X(1:end,1:end-1)'*y; % the new residual is calculated

   SS_LastParameter = SSRegression-SSRegressionPrevious;  % calculating the error improvement
   F = SS_LastParameter/(sigma*sigma);                    % compare the relative error
                                                          % improvement to the variance

   if(F<0.6)                           % if this ratio is smaller than a certain value the inclusion
                                       % of key samples will be stopped. See formula (3.35).

      'Number of keysamples is'        % when the inclusion of key samples is stopped the
      sX(2)                            % output is given as the number of key samples
      break                            % break out of the while-loop
```

```
    end

    SSRegressionPrevious = SSRegression; % when the loop is continued the current residual is
                                         % stored as the old residual
    sX = size(X);                        % storing the size of X

end                                      % end of while-loop
```

Plotting results

```
plot(x,y,'r.');                          % plotting of training set
hold on                                  % hold graphical result so that the next result can be
                                         % plotted in the same figure
plot(x,X(:,1:end-1)*b);                  % plot approximation
hold off
```

Function kernel

```
function output = kernel( x, xi)         % function declaration, the kernel used here is a
                                         % 1st order B-spline

if x-xi > 1 || x-xi < -1                 % if distance between input sample and key sample is
      output = 0;                        % more than 1 the output = 0
end
if x-xi <= 1 && x-xi>= 0                 % if (sample-key sample) is between 0 and 1 then the
      output = xi-x+1;                   % output will be a value on the declining B-spline slope
end
if x-xi < 0 && x-xi >= -1                % if (sample-key sample) is between –1 and 0 then the
      output = -xi+x+1;                  % output will be a value on the inclining B-spline slope
end
```

Function Aug(ment)

```
function [Xa,Ra] = aug( X, R, x)         % function declaration
sX = size(X);                            % determining size of input matrix X
sR = size(R);                            % determining size of input matrix R

R_bar = R(1:end-1,1:end-1);              % declaration of R̄ of formula (3.33a)
g = (inv(R(1:end-1,1:end-1)))'*X(1:end,1:end-1)'*x;   % calculation of g from (3.33b)

r_bar = R(1:end-1,end);                  % declaration of r̄ from (3.33.c)
gamma= sqrt((x'*x)-(g'*g));              % calculation of γ from (3.33d)
rho_bar= (1/gamma)*((x'*X(:,end))-(g'*r_bar));   % calculation of ρ̄ from (3.33e)
tau = sqrt((X(:,end)'*X(:,end))-(r_bar'*r_bar)-(rho_bar)^2);   % calculation of τ from (3.33f)

Xa = [X(:,1:end-1), x, X(:,end)];        % Augmenting X with vector x
                                         % putting R_a together
Ra = [R_bar g r_bar; zeros(1,length(R_bar)) gamma rho_bar; zeros(1,length(R_bar)) 0 tau];
```

Appendix C – Online KSM Matlab code

Main Code

Initialization

```
clear all;                               % clear memory
```

```
nrofommisions =0;                          % declaration of the variable which counts the numer
                                           % of omissions
ommissioncounter= 5;                       % declaration of the counter which counts back so after
                                           % 5 steps another omission attempt is undertaken

% input parameters

sigma_est= sqrt(0.005);                    % estimated noise value
maxerr_inc= 1.645;                 % tuning parameter for inclusion of potential key samples
maxerr_exc=1.3;                            % tuning prameter  for omission of key samples

% preset paramaters

omission_flag = 0;
counter = 20;
checknouw = 0;

%function initialization
n=100;                                     % number of training samples
sigma = sqrt(0.005);                       % actual noise value

v = 0.05;                                  % parameter used in target function
x = linspace(0,1,n)';                      % create input space vector x
sx = size(x);                              % determine size of x

x_func=x;                                  % store original x in x_func

y_func=sin(1./(x+v));                      % target function without noise
y = sin(1./(x+v))+sigma*randn(n,1);        % target function with noise
%y_func= sin(2*x).^2+cos(10*x);            % other possible target functions
%y = sin(2*x).^2+cos(10*x)+sigma*randn(n,1);
```

Randomizing

```
% Because samples will in practice be presented to the
% algorithm in a more or less random order, the samples % in the training set will here be randomized.

rand_x_index = randperm(n)';               % create a randomized index vector with size n
srand_x_index = size(rand_x_index);
for i = 1:srand_x_index(1)
   if rand_x_index(i)==0                   % randomizing index 0 is excluded because the training
                                           % set vectors have indices 1 to n
     rand_x_index(i)=1;
   end
   rand_x(i)=x(rand_x_index(i));           % randomizing the input space vector of the training set
   rand_y(i)=y(rand_x_index(i));           % randomizing the output space vector of the training set
end
x=rand_x';                                 % setting the randimized vectors in the wright form
y=rand_y';
sx= size(x);

%initialisation
nks=1;                                             % number of key samples is 1
xks=x(1:nks,:);                            % the first sample becomes a key sample for certain
Xks=[];                                            %  declaration of X_KeySample
for i = 1:nks
   for r= 1:nks
      Xks(i,r)=kernel(x(i,:),xks(r));              % fill indicator matrix X_ks with key samples
   end
end
```

RKSM initializing

```
R = sigma*[sqrt(Xks.^2) y(1)./sqrt(Xks.^2)];        % initializing matrix R by normalizing it for X_ks
Rks=chol(Xks);                                      % R_ks is cholesky decomposition of X_ks

Vinv=Rks'*inv(Rks)'*R(:,1:end-1)'*R(:,1:end-1)*inv(Rks)'*Rks';
                                                    % this is equal to sigma*sigma when having one
                                                    % keysample;

init_ks=nks;                                        % number of keysamples used in initialisation
for =init_nks:n
  %incoming sample
  for i = 1:nks
  f(i)=kernel(x(d,:),xks(i,:));                     % generate vector f from (3.51)
  end
   f=f(1:i);
```

Fine-Tuning

```
[R,rem_err]=finetune(R,sigma*[f y(d)]);             % fine-tuning of matrix R
```

Inclusion

```
incl_flag=0;                                        %set inclusiong flag

   if (sqrt(abs(rem_err)) > maxerr_inc*sigma_est)   % if remaining error is too large then
                                                    % include sample as key sample

                                                    % calculate update parameters for Rks
   theta= kernel(x(d,:),x(d,:));
   R12ks=Rks'\f';                                   % See formula (3.63a)
   R22ks=sqrt(max(0,theta-R12ks'*R12ks));           % See formula (3.63b)

   % check wether updated Rks will be stable
   if (R22ks > 2e-16);                              % check whether inclusion will be stable

      % calculate update parameters for R
      R12 = R(:,1:end-1)'\(Rks'*Rks*Vinv*f' + sigma^2*(f'*theta));   % See formula (3.60a)
                                % last term is multiplied bysigma^2 because values in R are normalized
                                                                     % for sigma
      R22 = sqrt(max(0, f*Vinv*f' + theta^2*sigma^2 - R12'*R12));   % See formula (3.60b)

      % check wether updated R will be stable
      if (R22 > 2e-15)

         %Update R, Rks and Vinv
         R   = [ R(:,1:end-1) R12 R(:,end); zeros(1,nks) R22 rem_err]; % put updated R together
         Rks = [ Rks R12ks; zeros(1,nks) R22ks];                       % put updated Rks together
         Vinv= [ Vinv zeros(nks,1); zeros(1,nks) sigma^2];             % weight matrix update
         nks = nks + 1;                                                % increase number of ks

         xks = [xks; x(d,:)];                                          % add key sample to existing set

         incl_flag = 1;
      end

   end
end
% if sample is not included as a key sample then finetune Vinv
```

```
if incl_flag==0
   l = (f/Rks)/Rks';                          % See formula (3.53a)
   Vinv = Vinv + sigma^2*l'*l;                % See formula (3.53b)
end
```

Omission

```
if (nks>5)                                    % start omission only when number of ks > 5
   if ommissioncounter==0;                    % check for omission candidates after 5 loops

    b = R(:,1:end-1)\R(:,end);                % calculating weights as done in the offline algorithm
    bVinv = diag(Vinv).*b;                    % see formula (3.66)
    [valuevect, indexvect] =sort(abs(bVinv(1:end-1)));
    minindex = indexvect(1);


    Rold=R;                                   % store R matrix
    R = QRswapcolumn(R,minindex);             % swap columns see formula (3.67)-(3.69)

    if (sqrt(abs(R(end,end)))< maxerr_exc*sigma_est)   % if the increase in error is smaller than
                                              % threshold than start omission

      nrofommisions=nrofommisions + 1;        % increase omission counter

      %change in sets
      xks_temp=xks(minindex,:);               % swap key sample set
      xks(minindex,:) = xks(end,:);
      xks(end,:) = xks_temp;

      Rks=QRswapcolumnRks(Rks,minindex);      % swapcolumns of matrix R_ks

      % changing Vinv

      Vinv= SwapcolumnVinv(Vinv,minindex);    % updating Vinv see formula (3.70)

      % storing info before actual ommison

      fullXks = Rks'*Rks;                     % storing values for updating Vinv
      cxks = fullXks(1:end-1,end);
      cl = Vinv(1:end-1,end);
      clambda = Vinv(end,end);

                                              % actual ommision

      Vinv(:,end) = [];
      Vinv(end,:) = [];

      R(:,end-1) = [];
      R(end,:) = [];

      xks(end,:) = [];
      nks = nks - 1;
      Rks = Rks(1:end-1,1:end-1);

                                              % updating Vinv after ommision

      cy = (Rks\(Rks'\cxks));                 % See formula (3.75b)
      Vinv = Vinv + cy*cl'+cl*cy' + cy*clambda*cy';   % See formula (3.75b)
```

```
        ommissioncounter = 5;                    % reset counter

      else
        R=Rold;                                  % when no omission is executed R keeps it old values
        ommissioncounter = 5;
      end
    else
      ommissioncounter = ommissioncounter - 1;
    end
    end
```

Plotting

```
   if size(xks,2) == 1

      % caculate b
      b = R(:,1:end-1)\R(:,end);       % calculate weights

      % plot
      xplot = linspace(0,1,1000)';   % generating input samples for plotting
      Xplot = [];
      for i = 1:1000
         for r= 1:nks
            Xplot(i,r)=kernel(xplot(i,:),xks(r));
         end
      end

      plot(x,y,'.', xplot, Xplot*b,'r');  % calculate and plot output values

      title(['sample number is ', num2str(d), ', number of ks is ', num2str(nks),' and number ofomissions is ',
num2str(nrofommisions)]);

      hold on;

      Xplot = [];
      for i = 1:length(xks)
         for r= 1:nks
            Xplot(i,r)=kernel(xks(i,:),xks(r));
         end
      end
      plot(x(d),y(d),'.r');
      plot(xks,Xplot*b,'o');  % plot key samples as “o”
      plot(x_func,y_func);
   end
hold off
pause(0.0000000001);              % pause needed for showing animation otherwise only the end
                                  % result will be displayed
end
```

function finetune

```
function [R, last] = finetune(R,x)          % function decleration

p = length(x);                              % determining length of newly added row

for k = 1:p-1
   [c, s, nu] = rotgen(R(k,k),x(k));        % First step of Givens rotation see appendix A
    R(k,k) = nu;
```

```
   x(k) = 0;
   [R1, R2] = rotapp(c,s,R(k,k+1:end),x(k+1:end));   % calculating  row values on the right side of index
                                                     % column
   R(k,k+1:end) = R1;                                % replacing column with newly calculated column
   x(k+1:end) = R2;
end
last = x(end)
```

function Qrswapcolumn

```
function R = QRswapcolumn(R,minindex);
% Swaps the columns minindex and last column of non-augmented R;
% In this R is given as:
%
% A = Q*R;
%
% So after the swapping, the system is retriangulised
% The matrix Q is not used.
%
% More information: Matrix algorithms (Stewart) pg. 334
%
% In this version the last row of R is not present, while in QRswapcolumn_ it is.

m = size(R,1);                                       % determine the size of R
R = [R;zeros(1,m) 100];                              % add one row with zero's to make it square
                                                     % because this matrix is augmented

mini = minindex;
maxi = m;

%swap column l and m
temp = R(:,mini);                                    % the last column is swapped with the index
R(:,mini) = R(:,maxi);                               % column
R(:,maxi) = temp;

                                                     % remove the spikes
for k = maxi-1:-1:mini+1                             % Matrix R is not upper-triangular after the
   [c,s,nu] = rotgen(R(k,mini),R(k+1,mini));         % the swapping of the columns. By means of
   R(k,mini) = nu;                                   % Givens rotation it brought back to its upper-
   R(k+1,mini) = 0;                                  % triangular form.
   [R1,R2] = rotapp(c,s,R(k,k:end),R(k+1,k:end));
   R(k,k:end) = R1;
   R(k+1,k:end) = R2;
end;

                                                     %remove the lowerdiagonal
for k = mini:maxi%-1                                 % the spike has been removes but the lower
   [c,s,nu] = rotgen(R(k,k),R(k+1,k));               % diagonal still contains non-zero elements
   R(k,k) = nu;                                      % which are also by Givens rotation removed
   R(k+1,k) = 0;
   [R1,R2] = rotapp(c,s,R(k,k+1:end),R(k+1,k+1:end));
   R(k,k+1:end) = R1;
   R(k+1,k+1:end) = R2;
end
R(end,:)=[];
```

Function SwapcolumnVinv

```
function Vinv = SwapcolumnVinv(Vinv,index)           % function declaration
```

```
index_col1=Vinv(1:index-1,index);                % storing original columns
index_col2=Vinv(index,index);                    % storing original columns
index_col3=Vinv(index+1:end-1,index);            % storing original columns
index_col4=Vinv(end,index);                      % storing original columns

end_col1=Vinv(1:index-1,end);                    % storing original columns
end_col2=Vinv(index,end);                        % storing original columns
end_col3=Vinv(index+1:end-1,end);                % storing original columns
end_col4=Vinv(end,end);                          % storing original columns

Vinv = [Vinv(1:index-1,1:index-1) end_col1 Vinv(1:index-1,index+1:end-1) index_col1;
      end_col1' end_col4' end_col3' end_col2';
      Vinv(index+1:end-1,1:index-1) end_col3 Vinv(index+1:end-1,index+1:end-1) index_col3;
      index_col1' index_col4' index_col3' index_col2'];

% here Vinv is updated by swapping the columns. The exact procedure is described in (3.70)
```

Appendix D – Blocked KSM Matlab code

Main Code

```
clear all;                                       % clear memory

NrofSamples = 10000;                             % Nr of samples in training set
blocksize   = 50;                                % Nr of samples in one block
NrofBlocks  = round(NrofSamples/blocksize);      % Total nr of blocks
sigma = 0.05;                                    % estimated noiselevel
noise = 0.05;                                    % noiselevel
xplot = linspace(0,1,250)';                      % used to plot the approximation
nrofomissions=0;                                 % decleration of variables
nrofloops=0;
percentage=0;

Pinv = [];
xks  = [];                                       % stored column-wise
yks  = [];
Q    = [];
Ra   = [];
nks  = 0 ;
X    = [];
Xks  = [];

for i = 1 : NrofBlocks                           % loop until all blocks have passed

   rugulactive = 0;
   newdata = 1;
   wrong_indicator = 0;

   if i==1                                       % create non randomized function values
      xdata=linspace(0,1,250);
      sxdata=size(xdata);
      for d = 1:sxdata(2);
         ydata(d)=sin(1./(xdata(d)+0.05));
      end
   end

   xn = rand(blocksize,1);                       % randomized training set samples
   yn = sin(1./(xn+0.05))+sigma*randn(blocksize,1);   % randomized training set samples
```

```
Xn= fillkernel(xn,xks);                 % Create matrix Xn with new samples in
                                        % the existing feature space

                                        % creating complete indicator matrix
X = [Pinv*Xks; Xn];                     % see formula (4.3)

% creating Xpotential
Xpot = fillkernel([xks; xn], xn);       % see formula (4.7)
Xpot(1:nks,:) = Pinv*Xpot(1:nks,:);     % weighting the existing key samples
y = [Pinv*yks;yn];                      % creating a vector with all sample output
                                        % values. The output values of the key sample
                                        % must be weighted

% calculate columnnorm
for j = 1:blocksize
   colnorm(j,1)=Xpot(:,j)'*Xpot(:,j);   % calculate the norm of each column of Xpotential
end

% incorporate the new info in the decomposition;

Ra = chol(Ra'*Ra + [Xn yn]'*[Xn yn]);   % finetune the augmented matrix R
if (nks)
   Q =X/Ra(1:nks,1:nks);                % results in Q without 0-space column vectors
   Q = reorthogonalise(Q);              % make Q orthonormal to computer precision
end

                                        % calculate the weighted residual
if (nks)
   if (newdata)
      ypr = Q*Ra(1:nks,end);
      e = y - ypr;
   else
      e = e - Q(:,nks)*Ra(nks,end);
   end
else
   e=y;
end
SS = e'*e;                              % size of error
```

Loop until noise is fitted

```
breaksignal=1;
while (breaksignal)
   nrofloops=nrofloops+1;
   % Select best potential indicator
   inpr = Xpot'*e;
   angle = inpr.*inpr./(colnorm*SS);    % see formula (3.10)
   [maxv,maxi]= max(angle);

   % include in current decompositions
   if (nks)
      if (min(abs(xks - xn(maxi)))< 1e-10)   % determinse the minimal width between
                                             % the keysamples
         disp('too close to present key sample')
         wrong_indicator=1;
         break;
      end;

      %1) regress new indicator on space spanned by present key samples
```

```
   rn = Q'*Xpot(:,maxi);                          % see Figure 4.2

   %2) generate new column of Q
   qn = Xpot(:,maxi) - Q*rn;                      % see Figure 4.2

   %3) determine amount of info in new indicator
   gamma = norm(qn,2);                            % if vector qn is to small the feature
                                                  % space is not extended well enough
                                                  % therefore the sample not included
                                                  % as a key sample
   if (gamma < 0.005*sigma)
      disp('new indicator contains not enough information')
      wrong_indicator=1;
      break;
   end

   %4) normalise the new column
   qn = qn/gamma;

   %5) calculate the projection of y on new indicator
   rho = qn'*e;

   %6) Put it all together
   Ra = [Ra(:,1:nks),[rn;gamma],[Ra(1:nks,end);rho]; zeros(1,nks+1) 1];
   Q  = reorthogonaliseColumn(Q,qn);

else

   Ra = chol([Xpot(:,maxi) y]'*[Xpot(:,maxi) y]);  % if there are no key samples yet Ra and Q
   qn = Xpot(:,maxi)/(norm(Xpot(:,maxi)));         % are initialized here
   Q  = qn;
end

 if(wrong_indicator)                              % if sample contained not enough information
   break                                          % then break loop
 end

 X =[X Xpot(:,maxi)];                             % Include new indicator

 xks = [xks; xn(maxi)];                           % new key samples
 nks = nks +1;

 e = e - Q(:,nks)*Ra(nks,end);                    % calculate the difference in the size of error
 SSold = SS;
 SS = e'*e;

 %prevent double inclusion
 colnorm(maxi) = inf;
 newdata  = 1;
 if((SSold-SS)/(noise)^2 < 5)                     % see formula (3.35)
   X(:,end) = [];
   Ra(:,nks)= [];
   Ra(end,:)= [];
   Q(:,end) = [];
   xks(end,:)=[];
   nks = nks - 1;
   breaksignal = 0;

   disp('loop ended')
   percentage = (nrofloops/blocksize)*100;
```

```
    end
    if (min(colnorm) == inf)
      disp('all samples included');
      breaksignal = 0;
    end
 end

Xks= fillkernel(xks,xks);                    % fill the keysample matrix

Pinv = Ra(1:nks,1:nks)/Xks;                  % update the weight matrix
Vinv = Pinv'*Pinv;
[q,Pinv]= qr(Pinv);
```

Omission

```
                                             % the omission is executed nearly the same as the in
                                             % the online KSM algorithm therefore one the
                                             % the differences have been commented
for q= 1:nrofloops

   if (nks>5)

    b = Ra(:,1:end-1)\Ra(:,end);
    bVinv = diag(Vinv).*b;
    [valuevect, indexvect] =sort(abs(bVinv(1:end-1)));
    minindex = indexvect(1);

    Rold=Ra;
    Ra = QRswapcolumn(Ra,minindex);

      if (sqrt(abs(Ra(end,end)))< 4.5*sigma)

        nrofomissions=nrofomissions + 1;

        % change in sets
        xks_temp=xks(minindex,:);
        xks(minindex,:) = xks(end,:);
        xks(end,:) = xks_temp;

        % change X and Xks
        Xtemp = X(:,end);
        X(:,end)=X(:,minindex);
        X(:,minindex)=Xtemp;

        % actual ommision

        X(:,end)=[];

        Ra(:,end-1) = [];
        Ra(end,:) = [];

        xks(end,:) = [];
        nks = nks - 1;

        Xks= fillkernel(xks,xks);

        % updating Vinv after ommision
        Pinv = Ra(1:nks,1:nks)/Xks;          % updating the weight matrices
        Vinv = Pinv'*Pinv;
        [q,Pinv]= qr(Pinv);
```

```
            % updating Q

            Q = X/Ra(1:nks,1:nks);          % results in Q without 0-space column vectors
            Q = reorthogonalise(Q);         % orthonormalize Q to computer precision

          else
             Ra=Rold;
          end
       end
    end

    nrofloops=0;

    b= Ra(1:nks,1:nks)\Ra(1:nks,end);
    yks = Xks*b;

    plot(xn,yn,'.');
    title(['block number is ', num2str(i), ', number of ks is ', num2str(nks),' and number of omissionsis ',
num2str(nrofomissions),'      percentage of samples used for inclusion ' num2str(ercentage) ]);
            hold on
            Xplot = fillkernel(xplot, xks);
            for p = 1:nks
       plot(xks(p),yks(p),'r.','MarkerSize',5*sqrt(Vinv(p,p)));
    end
    plot(xplot, Xplot*b);
    plot(xdata, ydata, 'r');
    hold off
    axis([0 1 -1.2 1.2])
    drawnow
end
```

8 Literature

Kruif, B.J. de, (2004), “Function Approximation for Learning Control”, PhD. Thesis. University of Twente, Enschede

Stewart, G,W. (1998), “Matrix Algorithms”, vol 1: Basic decompositions. SIAM, Philidelphia.

Controllab Products B.V., (2005) (http://www.20sim.com/)

Kruif, B.J. de, (2005), “Implementation of BKSM” (confidential), Imotec B.V., Hengelo (ov)

Golub, G. H. and Van Loan, C., (1996), “Matrix computations”, The Johns Hopkins University Press, Baltimore, Maryland, 3rd edn..

Dirne, H. (2005), “Demonstrator of advanced controllers”, M. Sc. Thesis, University Twente

Vries, T.J.A. de, (2003), course book “Mechatronic Design of Motion Systems”. University of Twente.

Amerongen, J. van, (2001), course book “Regeltechniek”, University of Twente, Enschede