LIDS-P-2067

Inverse Passive Learning of an Input-Output-Map Through Update-Spline-Smoothing

M. Heiss
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

September 5, 1991

Abstract

This paper presents a robust method of learning passively a one-dimensional input-output-map when receiving only indirect information about the correct input-output-map (e.g. only the sign of the deviation between the actual estimated output value and the correct output value is obtained). This information is obtained for only one input-output combination per updating-cycle.

The approach is to increment or decrement step by step the output values of the actually stored map and then to apply global or local cubic spline smoothing, in order to avoid "adaptation holes" at points which are never updated or less frequently updated than other points. This method works with noisy measurements as well as slowly time-varying systems. Even discontinuous changes of the desired input-output-relation do not result in instability.

Problems of convergence and stability are treated and design rules are given.

Address for correspondence **until September 30, 1991**:
Dr.Michael Heiss
LIDS/M.I.T. 35-417
Cambridge, MA-02139-4307
U.S.A.

Address for correspondence **after Oktober 1, 1991**:
Dr.Michael Heiss
R.Waisenhorngasse 115/52
A-1235 Vienna, Austria
EUROPE
Tel.: 011 (431) 883174, FAX: 011 (431) 883174-90

Inverse Passive Learning of an Input-Output-Map Through Update-Spline-Smoothing*

M. Heiss
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

September 5, 1991

Abstract

This paper presents a robust method of learning passively a one-dimensional input-output-map when receiving only indirect information about the correct input-output-map (e.g. only the sign of the deviation between the actual estimated output value and the correct output value is obtained). This information is obtained for only one input-output combination per updating-cycle.

The approach is to increment or decrement step by step the output values of the actually stored map and then to apply global or local cubic spline smoothing, in order to avoid "adaptation holes" at points which are never updated or less frequently updated than other points. This method works with noisy measurements as well as slowly time-varying systems. Even discontinuous changes of the desired input-output-relation do not result in instability.

Problems of convergence and stability are treated and design rules are given.

1 Introduction

The term *learning* is used in a large variety of meanings. The work reported in this paper deals with the learning of an input-output-map, having m inputs $\boldsymbol{\xi} = (\xi_1 \dots \xi_m)^T$ and one output $f(\boldsymbol{\xi})$. In other words an m-dimensional hypersurface $f(\boldsymbol{\xi})$ has to be learned. To specify the term *learning* more clearly in this context, a distinction should be made between passive and active learning. Passive learning is defined as learning in the sense that the learner pays no role in obtaining information about the unknown input-output-map, as opposed to active learning where the learner has complete choice in the information received.

*This work was supported by the Austrian "Fonds zur Förderung der wissenschaftlichen Forschung" under contract J0514TEC.

In the most widely studied *passive learning of an input-output-map*, learning simply means collecting examples, consisting of the input vectors ξ_i and the corresponding output values at those locations, i.e. the heights of the surface $f(\xi_i)$ [1]. Most of the methods allow noisy measurements of $f(\xi_i)$ [2, 3, 4, 5, 6, 7, 8, 9]. The problem solved in these papers is to reconstruct the hypersurface from the set of sparse data points. The related keywords for this kind of problem are regularization theory, approximation, estimation, smoothing, regression, and interpolation [10, 11, 12]. In particular for high dimensions m of the input vector ξ the approximation problem is hard to solve [13, 4, 7, 14, 15]. This effect is the so called "curse of dimensionality" (Bellman 1961 [16]).

In applications where the input-output-map is doing the work of a controller the *inverse* problem of how to choose the input vector ξ to get a certain output value $f(\xi)$ has to be solved (Atkeson 1986-91 [17, 7, 6, 18, 19]) [20].

In reflection of a wide range of applications the following problem is posed: Let $\xi = (\xi_M, u)^T$ be composed of an $(m-1)$-dimensional vector ξ_M, representing the given measured input variables, and u, the scalar control variable. Define $\mathcal{F} \subset \mathbf{R}, \mathcal{X}_M \subset \mathbf{R}^{m-1}, \mathcal{U} \subset \mathbf{R}$ as the application specific set of all possible values of $f(\xi), \xi_M$, and u, respectively. Let $f(\xi_M, u) : \mathcal{X}_M, \mathcal{U} \rightarrow \mathcal{F}$ be "practically invertible" in the sense that for every combination of elements of $\mathcal{F}$ and $\mathcal{X}_M$, denoted f_q[1] and ξ_{Mq} exists a $u_q \in \mathcal{U}$ such that $f(\xi_{Mq}, u_q) = f_q$ (see Fig.1).

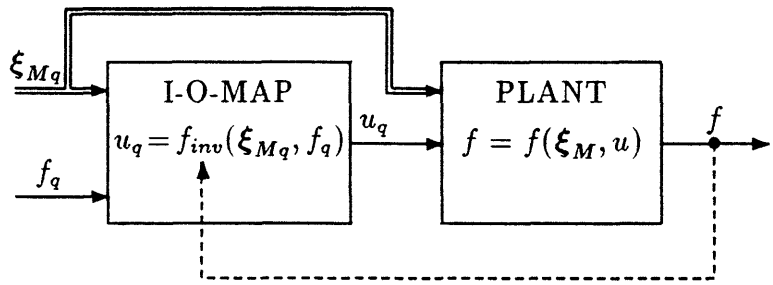

Figure 1: Nonlinear plant and input-output-map working as controller: the desired $f = f_q$ is obtained by looking up the control variable u_q in the correctly learned input-output-map

Then we can define the inverse function of

$$f = f(\xi_M, u) : \mathcal{X}_M, \mathcal{U} \rightarrow \mathcal{F} \tag{1}$$

as

$$u = f_{inv}(\xi_M, f) : \mathcal{X}_M, \mathcal{F} \rightarrow \mathcal{U}. \tag{2}$$

It is assumed that a very basic relationship between the change of the control variable u and the plant output f is known, by knowing a constant

[1]Index q for "query"

$$c_{sign} = sign(\frac{\partial f(\xi_M, u)}{\partial u}), \qquad \forall\, \xi_M \in \mathcal{X}_M, u \in \mathcal{U}. \tag{3}$$

The issue is to learn the inverse function $f_{inv}(\xi_M, f)$ with the following features:

1. the plant can be a black box system with the inputs ξ_M, u and the output f, describable by the unknown nonlinear function $f = f(\xi_M, u)$ and relation (3)
2. the function $f(\xi_M, u)$ and therefore $f_{inv}(\xi_M, f)$ may vary very slowly during the life of the plant (algorithm must forget the old experience),
3. the answer for every query $f_{inv}(\xi_{Mq}, f_q)$ must be found immediately by looking it up in an input-output-map, and no post-computations are allowed,
4. $f_{inv}(\xi_{Mq}, f_q)$ is known to be smooth; thus every estimate of f_{inv} must be smooth, too,
5. the learner plays no role in obtaining information about the unknown input-output-map (passive learning),
6. robust for noisy measurements,
7. arbitrary accuracy of u,
8. easily realizable.

2 Project Motivation

For a better understanding of the posed problem, an academic example is given, only to demonstrate the context and not to be considered as a real problem: *The Golden Toast Problem.*

Let us assume you want to make a golden toast every day in the morning. Eq. (1) is the toaster model. In this case the color of the toast, the desired output of the plant, $f = f_{gold}$ is constant. To make it easy, the other input variables ξ_M are reduced to only one scalar $\xi_M = \vartheta_{bread} \in [-40°C, +30°C]$, the bread temperature at the time of putting the toast into the toaster. Depending on the bread temperature ϑ_{bread} and the control knob position *ckp* you will produce a toast with the color

$$f = f(\vartheta_{bread}, ckp) = f(\xi_M, u). \tag{4}$$

If you own a toaster which is powerful enough to produce a golden toast even if you take the bread out of a deep freezer with $-40°C$, the function f is *practically invertible* and you have to learn according to eq. (2)

$$ckp = f_{inv}(\vartheta_{bread}, f_{gold}). \tag{5}$$

Because of the constant setting $f = f_{gold}$, the dimension of $f_{inv} : \mathbf{R}^2 \rightarrow \mathbf{R}$ can be reduced to $\tilde{f}_{inv} : \mathbf{R} \rightarrow \mathbf{R}$, by writing

$$ckp = \tilde{f}_{inv}(\vartheta_{bread}). \tag{6}$$

It is known that a higher *ckp* leads to a darker toast (higher f). Thus, c_{sign} of eq. (3) is $+1$.

Every morning you can measure the bread temperature ϑ_{bread}. Then you look up in your mental input-output-map the necessary control knob position *ckp* to produce a golden toast. If the toast is too dark, you will decrement the control knob position for this specific temperature in your mental map. If the toast is too light, you will increment it. After a couple of days you will have no problems to make your toast as you like it. Doing this every day, even a slowly drifting toast characteristic would not handicap you in producing perfect toasts.

As long as you do not have a sensor available which is able to measure the plant output f during the process or is not fast enough to be used for an on-line feedback-control you have to learn the relationship (2),(6). With this relationship a fast on-line feed forward control can be done, using the updating procedure as slow feedback-loop. In some applications the input-output-map is just used for a fast but rough inner control loop and additionally an accurate outer control loop is provided. Note that in most cases passive learning is required. No test series are allowed.

Similar problems can be found in steel production and robot control. Numerous applications can be found in automotive control:

- single cylinder control for modern Diesel engines, compensating for the tolerances of the different cylinders over rotation speed and injection quantity (in this case z input-output-maps are necessary for z cylinders) (Heiss and Augesky 1988 [21, 22]),
- adaptive full-load fuel control to avoid too high injection quantities which would result in bad smoke emission values (Heiss 1989 [21, 23]),
- adaptation of ignition angle for different fuel qualities (Kiencke 1987 [24]),
- control of knock frequency with forward-loop adaptation (Kiencke 1987 [24]),
- line pressure control for automatic transmission (Yamaguchi 1990 [25]),
- position control during friction (Bitzer 1989 [26], Silberbauer 1990 [27]),
- and many others [28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44].

3 Update-Smoothing or Query-Smoothing

The one-dimensional case ($m = 1$) is very important for industrial applications due to the possibility of cheap realization. Even this one-dimensional case has not been solved in general up to now [see above mentioned application papers]. Therefore, in this first paper only the one-dimensional case of inverse passive learning of the function

$$u = f_{inv}(x) \qquad \forall\, x \in \mathcal{X} = [a, b] \tag{7}$$

will be treated, where u and x are scalars (e.g.: ckp and ϑ_{bread}).

The function f_{inv} is known to be smooth. Then for every given accuracy Δu_{max} (see feature 7) a constant sampling interval Δx can be found (e.g. $\Delta x = \frac{\Delta u_{max}}{\max_{x \in \mathcal{X}} \frac{df_{inv}}{dx}}$) such that

$$f_{inv}(x) - f_{inv}(i\Delta x + c) \leq \Delta u_{max} \qquad \forall\, x \in \mathcal{X}, \tag{8}$$

where $i = \text{argmin}_j\, |x - j\Delta x - c|$ and c is a constant which makes $i(x = a) = 1$. With $\mathbf{x} = (x_1, x_2 \ldots x_n)^T$, $x_i = i\Delta x + c$, and $\mathbf{u} = (u_1, u_2 \ldots u_n)^T, u_i = f_{inv}(x_i)$, the problem is reduced to learning an input-output-mapping

$$\mathbf{x} \rightarrow \mathbf{u}, \tag{9}$$

where $\mathbf{x}$ is constant and known, and $\mathbf{u}$ is very slowly time varying and unknown (Fig.2). For simplicity $\mathbf{u}$ can be treated as being constant over time.[2]

At the very beginning instead of the unknown $\mathbf{u}$ an initial estimate $\hat{\mathbf{u}}(0)$ is proposed. This initial estimate $\hat{\mathbf{u}}(0)$ may be any vector with $\hat{u}_i \in \mathcal{U}, \forall\, i \in \{1 \ldots n\}$ and might be a very bad estimate of $\mathbf{u}$. When the system starts running, the first query at x_j is answered with $\hat{u}_j(0)$. The control variable is used as input to the plant. After completing the process the output $f(0)$ of the plant is measured. If $f(0)$ is not equal to the desired output $f_d(0)$ then $\hat{u}_j$ is updated according to

$$\hat{u}_j(1) = \hat{u}_j(0) + c_{sign}\, h(f_d(0) - f(0)). \tag{10}$$

If nothing more than (3) is known about the system, then the most robust way of choosing the operator h is

$$h(\Delta) = c_{incr}\, sign(\Delta). \tag{11}$$

Otherwise $h(\Delta)$ can also be a gain or a more complicated controller. c_{incr} is the constant increment or decrement.

The simplest way of updating would be to update only $\hat{u}_j$ and to keep all the other $\hat{u}_i$ fixed:

$$\hat{u}_i(1) = \hat{u}_i(0) \qquad \forall\, i \in \{1 \ldots n\}, i \neq j. \tag{12}$$

[2]The algorithm, to be proposed in the following chapter, will work in the same way whether $\mathbf{u}$ is constant or very slowly time varying. Most of the other known algorithms do not share this property.

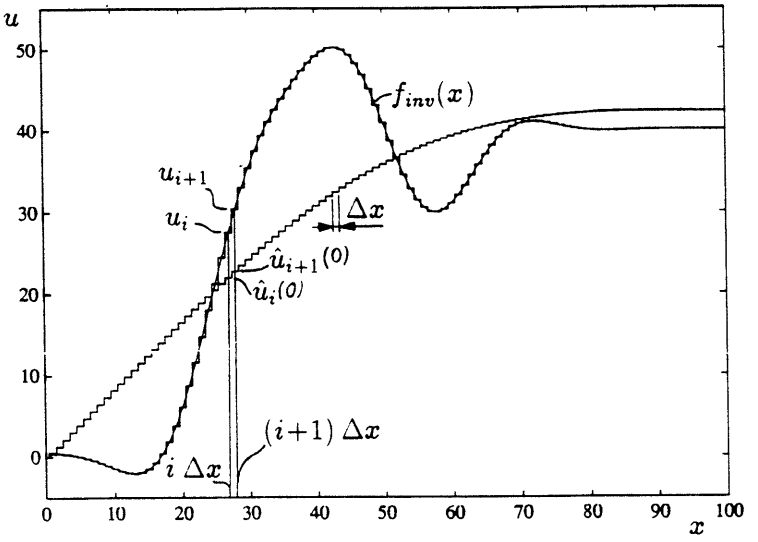

Figure 2: $f_{inv}(x)$, represented by the step function or the vector $\mathbf{u} = (u_1 \dots u_n)^T$, has to be learned. $\hat{\mathbf{u}}(0)$ is the initial estimate of $\mathbf{u}$.

Continuing to change only the data points where information is obtained, can cause problems when the number n of x-intervals is too high and/or it cannot be guaranteed that all $\hat{u}_i$ can be updated evenly. Some $\hat{u}_i$ may never be adapted and remain at their initial value $\hat{u}_i(0)$. Such non-adapted or less frequently adapted points $\hat{u}_i$ are so called "adaptation holes" [45] and are unacceptable (see feature 4). Every application has to deal with this problem.

Holmes 1989 [38] uses only a very small number n of intervals. Consequently the accuracy of $\mathbf{u}$ is low and the step height between u_i and u_{i+1} is high, but every $\hat{u}_i$ is well adapted. Holmes utilizes hysteresis on all the interval limits in order to hide the high steps of $\hat{\mathbf{u}}$ and to avoid an irritation of an outer-loop controller, which has the input-output-map in its control loop. This idea is applicable only for a very limited set of applications.

Schmidt and Schmitt 1988 [31] invented the "tent roof"-adaptation. They do not only modify the value $\hat{u}_j$ but also its $2r$ neighbors (r ... tent roof radius) in a decreasing way with increasing distance from x_j:

$$\hat{u}_{j+l}(k+1) = \hat{u}_{j+l}(k) + (1 - \frac{|l|}{r})\, c_{sign}\, sign(f_d - f(x_j(k))) \quad \forall\, l \in \mathbf{Z}, |l| \leq r. \qquad (13)$$

The disadvantage of this method is that a non-adapted or a less frequently adapted value $\hat{u}_j$ does not necessarily fit smoothly between its neighbors. The advantage of this method is that it can be extended easily to the multi-dimensional case.

The approach in this paper is to update $\hat{u}_j$ and then immediately smooth the entire vector $\hat{\mathbf{u}}$. In this case a smooth $\hat{\mathbf{u}}$ can be guaranteed at any time (see feature 4). The smoothing can be accomplished by multiplying $\hat{\mathbf{u}}$ with a constant smoothing matrix $\mathbf{S}$ (see section 5). This kind of smoothing is called *update-smoothing* because the smoothing is part of the updating procedure.

An alternative approach would be to collect the unsmoothed updates and to smooth the data only if a query is posed (Atkeson 1991 [19]). The latter can be called *query-smoothing.*

The proposed feature 3 requires the *update-smoothing* and prohibits the *query-smoothing.*

4 The Update-Smoothing Algorithm

According to the previous section the following algorithm is proposed for *inverse passive learning of an input-output-map through update-smoothing*:

- find an appropriate sampling interval Δx (8)
- find the constant c_{sign}, representing the relationship (3)
- find an initial estimate $\hat{\mathbf{u}}(0)$ which roughly estimates $\mathbf{u}$
- set the desired plant output f_d to a constant value or define $f_d = x$
- choose the controller function h (e.g.(11))
- compute the smoothing matrix $\mathbf{S}$ (18)
- start the system, $(k = 0)$
- repeat forever:
    - read query x_j
    - look up $\hat{u}_j(k)$ in the actual (k-th) input-output-map at x_j
    - use $\hat{u}_j(k)$ as input to the plant
    - wait for completion of the plant process
    - measure $f(k)$
    - set $\hat{u}_j(k+) = \hat{u}_j(k) + c_{sign}\, h(f_d - f(k))$ *** updating ***
    - set $\hat{\mathbf{u}}(k+1) = \mathbf{S}\, \hat{\mathbf{u}}(k+)$ *** smoothing ***
    - $k = k + 1$

The algorithm satisfies all the required features 1 to 7. In particular the combination of feature 2 (slowly time varying $\mathbf{u}$), feature 3 (immediate answer of queries), and feature 4 (smooth $\hat{\mathbf{u}}$) are not provided by other published methods. Feature 8 (easily realizable) will be shown in the next section.

5 Global Spline Smoothing

A well known technique for smoothing the data vector $\hat{\mathbf{u}}(k+)$ is to find

$$\hat{f}_{inv}(x) = \underset{g(x)}{\operatorname{argmin}} \left[\sum_{i=1}^{n} (\hat{u}_i(k+) - g(x_i))^2 + p \int_{-\infty}^{\infty} (g''(x))^2 dx \right], \tag{14}$$

optimizing $g(x)$ over the Sobolev space W_2 of functions with g' absolutely continuous and $g'' \in L_2$ (Schoenberg 1964 [46], Reinisch 1967 [47], DeBoor 1978 [48], Wahba 1975–1990 [3, 49], Craven and Wahba 1979 [50], Schumaker 1981 [51], Eubank 1984 [52, 53, 54], Wegman 1983 [5], Silverman 1985 [55]). p is a constant smoothing parameter trading off the smoothness of the curve $\hat{f}_{inv}(x)$ with its closeness to the values $\hat{\mathbf{u}}(k+)$ (the basic underlying idea was described by Whittaker 1923 [56]). The solution is known to be a cubic spline approximation of $\hat{\mathbf{u}}(k+)$ (Reinisch 1967 [47]). When $p = 0$, the solution is *any* interpolating function, $\lim_{p \to 0}$ of the solutions is the cubic spline interpolation, for $p \to \infty$ the solution is the least squares line (Buja 1989 [13], Wahba 1989 [57], Wahba 1990 [49] p.14).

With $\hat{u}_i(k+1) = \hat{f}_{inv}(x_i)$ it can be shown that (14) is equivalent to solving

$$\hat{\mathbf{u}}(k+1) = \underset{\hat{\mathbf{u}}}{\operatorname{argmin}} \, \|\hat{\mathbf{u}}(k+) - \hat{\mathbf{u}}\|_2^2 + p \, \hat{\mathbf{u}}^T \mathbf{K} \hat{\mathbf{u}}, \tag{15}$$

where

$$\mathbf{K} = \frac{6}{\Delta x^3} \underbrace{\begin{pmatrix} 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & 1 \; -2 \; 1 \end{pmatrix}^T}_{\text{dim: } n \times (n-2)} \cdot \underbrace{\begin{pmatrix} 2 & 1 & & & \\ 1 & 2 & 1 & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & 1 \; 2 & 1 \\ & & & & 1 \; 2 \end{pmatrix}^{-1}}_{\text{dim: } (n-2) \times (n-2)} \cdot \underbrace{\begin{pmatrix} 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & 1 \; -2 \; 1 \end{pmatrix}}_{\text{dim: } (n-2) \times n} \tag{16}$$

(Buja 1989[3] [13]). Note that $\mathbf{K}$ is of rank $n-2$. With $\mathbf{K} = \mathbf{K}^T$ and setting the derivative of the right hand term of (15) $\frac{\partial}{\partial \hat{\mathbf{u}}} = \mathbf{0}$ the solution of (15) is

$$\hat{\mathbf{u}}(k+1) = \mathbf{S} \, \hat{\mathbf{u}}(k+), \tag{17}$$

where

$$\mathbf{S} = (\mathbf{I} + p\mathbf{K})^{-1}. \tag{18}$$

Note that $\mathbf{K}$ is a constant matrix and so is $\mathbf{S}$. Thus, the smoothing procedure is a simple matrix multiplication (17) which is easily realizable.

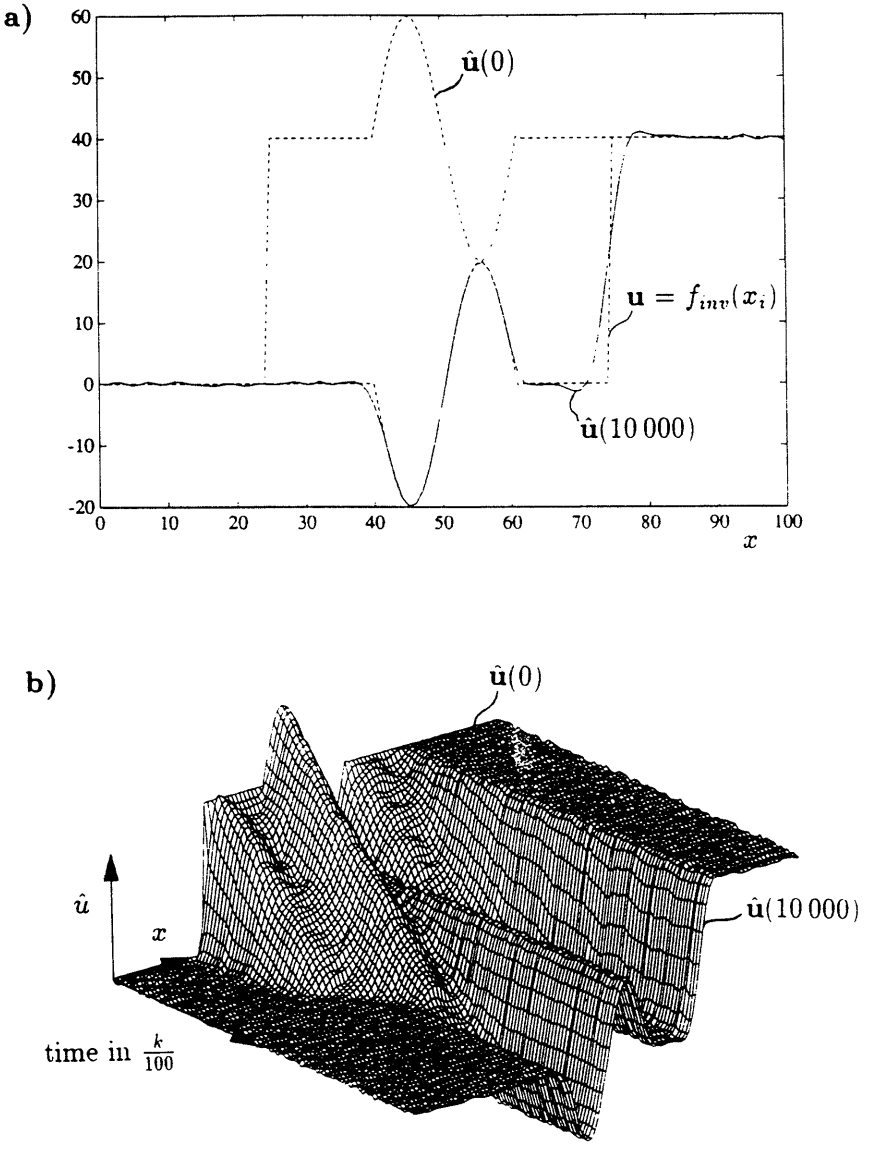

Figure 3: Simulation of how **u** is approximated when starting (k=0) with a completely different estimate $\hat{\mathbf{u}}(0)$ and $p = 0.0083$.

Example

Fig.3 shows how the algorithm works. $\mathbf{u} = f_{inv}(x_i)$ is the sampled unknown function to be found. $\hat{\mathbf{u}}(0)$ is an initial estimate of $\mathbf{u}$, being very different hereof. $f_{inv}(x)$ is chosen to be discontinuous at $x = 75$ and not smooth at $x = 40$ and $x = 60$ to show the properties of the algorithm. The vector $\mathbf{u}$ is defined to be smooth, if

$$\mathbf{u} \approx \mathbf{S}\,\mathbf{u}. \tag{19}$$

The algorithm does not know of the fact that $f_{inv}(x)$ is not a smooth function. Thus, the result of the learning process can only be a smooth approximation of $f_{inv}(x)$. The reason for this unusual choice of $\mathbf{u}$ is to give an example that even if $f_{inv}(x)$ is not as smooth as expected, the proposed algorithm does not become unstable.

The inputs x_j (e.g. ϑ_{bread}) of the map were given randomly. The output $\hat{u}_j(k)$ of the map was used as input to the plant. If the output f of the plant was higher than the desired value f_d, then $\hat{u}_j(k)$ was decremented ($\hat{u}_j(k+) = \hat{u}_j(k) - 1$), otherwise it was incremented. Then the smoothing $\hat{\mathbf{u}}(k+1) = \mathbf{S}\,\hat{\mathbf{u}}(k+)$ was done. Every $100k$ a new line $\hat{\mathbf{u}}$ was plotted in Fig.3b. Fig. 3a compares $\hat{\mathbf{u}}(10\,000)$ with $\mathbf{u}$ and shows that $\hat{\mathbf{u}}(10\,000)$ is a very good approximation of $\mathbf{u}$ as long as the requirement of a smooth f_{inv} is satisfied. The small limit cycle of $\hat{\mathbf{u}}$ around $\mathbf{u}$ is caused by the increment constant c_{incr}, eq. (11), being equal to 1.

6 Convergence and Stability

The example has shown that the algorithm works "pretty well". Nevertheless, it would be of interest to give at least some ideas of convergence and stability in general.

In this context *convergence* means that there exists an $K < \infty$ such that

$$|\hat{u}_i(K) - u_i| \leq c_{limit\ cycle} \qquad \forall\, i \in \{1 \ldots n\}, \tag{20}$$

where $c_{limit\ cycle}$ is a small constant satisfying the needs of the application. Usually $c_{limit\ cycle}$ is approximately c_{incr} (see eq. (11)). *Stability* means that eq. (20) holds for all $k \geq K$.

Referring to the algorithm, the two steps *updating* and *smoothing* are responsible for the change of $\hat{\mathbf{u}}$. The updating step always tries to improve the approximation of $\mathbf{u}$ as long as the controller h is chosen properly (10), (11). The critical step is the smoothing operation.

The global spline smoothing (14),(15) is only able to smooth the function and has originally nothing to do with estimating $\mathbf{u}$, except the fact that $\mathbf{u}$ is known to be smooth (19). Thus, $\hat{\mathbf{u}}$ will be smoothed even if $\hat{\mathbf{u}}$ is exactly $\mathbf{u}$. That means that stability is not obvious although the algorithm is convergent. Global spline

[3]The introduction of this paper, p.453-467, gives a very good survey of linear smoothers, in particular cubic spline smoothers.

smoothing can result in loss of stability, but smoothing is necessary to guarantee feature 4. Of course, eq. (19) says that the forgetting of an already learned $\hat{\mathbf{u}} = \mathbf{u}$ through smoothing is very slow.

In order to give a theoretical background to this problem some subproblems are considered:

1. What happens, if smoothing is applied infinitely often to $\hat{\mathbf{u}} = \mathbf{u}$ without updating in the meantime? (section 6.1)
2. How fast does the algorithm forget an already learned $\mathbf{u}$? (section 6.2)
3. If sufficient information is obtained for some of the n points of $\mathbf{u}$, to what values will the intermediate points converge? (section 6.3)

6.1 Infinitely many times of smoothing without updating

Without updating there is no longer a difference between k and $k+$ in eq. (17). $\hat{\mathbf{u}}(k)$ can be calculated immediately by $\hat{\mathbf{u}}(k) = \mathbf{S}^k\hat{\mathbf{u}}(0)$. For $k \rightarrow \infty$ we get $\hat{\mathbf{u}}(\infty) = \mathbf{S}^\infty\hat{\mathbf{u}}(0)$. What is $\hat{\mathbf{u}}(\infty)$ in relation to $\hat{\mathbf{u}}(0)$?

Theorem 1 *If* $\hat{\mathbf{u}}(0)$ *is smoothed infinitely many times by* $\hat{\mathbf{u}}(k+1) = \mathbf{S}\,\hat{\mathbf{u}}(k)$, *where* $\mathbf{S}$ *is defined in eq. (18), then* $\lim_{k\rightarrow\infty}\hat{\mathbf{u}}(k)$ *is the least squares regression line of* $\hat{\mathbf{u}}(0)$. *This means that*

$$\mathbf{S}^\infty = \mathbf{H}, \tag{21}$$

where $\mathbf{H}$ *is the hat matrix of the linear least squares regression.*

Proof: Since $\mathbf{S}$ is symmetric it can be diagonalized and the k-th power of $\mathbf{S}$ can be written as (Strang[4] [58] p.296, 264)

$$\mathbf{S}^k = \mathbf{V}\,\mathbf{\Lambda}^k\mathbf{V}^T = \lambda_1^k\,\mathbf{v}_1\mathbf{v}_1^T + \lambda_2^k\,\mathbf{v}_2\mathbf{v}_2^T + \lambda_3^k\,\mathbf{v}_3\mathbf{v}_3^T + \ldots\lambda_n^k\,\mathbf{v}_n\mathbf{v}_n^T. \tag{22}$$

$\{\lambda_i\}_{i=1}^n$ are the eigenvalues of $\mathbf{S}$ sorted to be of decreasing amount with increasing i, and $\{\mathbf{v}_i\}_{i=1}^n$ are the corresponding orthonormalized eigenvectors. $\mathbf{\Lambda}$ =diag$\{\lambda_i\}$ and $\mathbf{V} = (\mathbf{v})_{i=1}^n$.

It is known(e.g.[13] p.463) that $\lambda_1 = \lambda_2 = 1$ and $0 < \lambda_i < 1$ for $i \geq 3$. Due to this fact, (22) can be written as

$$\mathbf{S}^k = (\mathbf{v}_1\mathbf{v}_1^T + \mathbf{v}_2\mathbf{v}_2^T) + \lambda_3^k\,\mathbf{v}_3\mathbf{v}_3^T + \ldots\lambda_n^k\,\mathbf{v}_n\mathbf{v}_n^T. \tag{23}$$

The first term represents a projection on the plane spanned by $(\mathbf{v}_1, \mathbf{v}_2)$ ([58] p.380). The eigenvectors of $\lambda_1 = \lambda_2 = 1$ are defined by $\mathbf{S}\,\mathbf{v} = \mathbf{v}$, in other words the space of all $\mathbf{v}$ which are reproduced by spline smoothing. Since constants and straight lines

[4] a fascinating book about linear algebra

are not changed by spline smoothing ([13] p.461), the plane spanned by the vectors $(\mathbf{1}, \mathbf{x})$ is exactly the same plane as spanned by any other eigenvectors $(\mathbf{v}_1, \mathbf{v}_2)$ of $\lambda_1 = \lambda_2 = 1$. The projection matrix projecting on the plane $(\mathbf{1}, \mathbf{x})$ is known to be the least squares line hat matrix $\mathbf{H}$ ([58] p.158-160, [13] p.458). Thus, (23) becomes

$$\mathbf{S}^k = \mathbf{H} + \lambda_3^k \, \mathbf{v}_3 \mathbf{v}_3^T + \ldots \lambda_n^k \, \mathbf{v}_n \mathbf{v}_n^T. \tag{24}$$

For $k \to \infty$ all terms with $0 < \lambda_i < 1$ vanish, leaving $\mathbf{S}^\infty = \mathbf{H}$. □

6.2 Forgetting dynamics

The last section has shown that even an already learned $\hat{\mathbf{u}}$ will drift towards its regression line after many times of smoothing without updating (Fig.4). The same problem occurs when updating takes place for a long time only in a little local region of $\hat{\mathbf{u}}$ and not at the whole range of x_i-values. For example, updating the left endpoint of $\hat{\mathbf{u}}$ does not directly influence the right end of $\hat{\mathbf{u}}$ in the sense that it makes almost no difference for the right end of $\hat{\mathbf{u}}$ if the left end is not updated and the whole $\hat{\mathbf{u}}$ is only smoothed. Step by step, the right end will become smoother and smoother until it is a straight line.

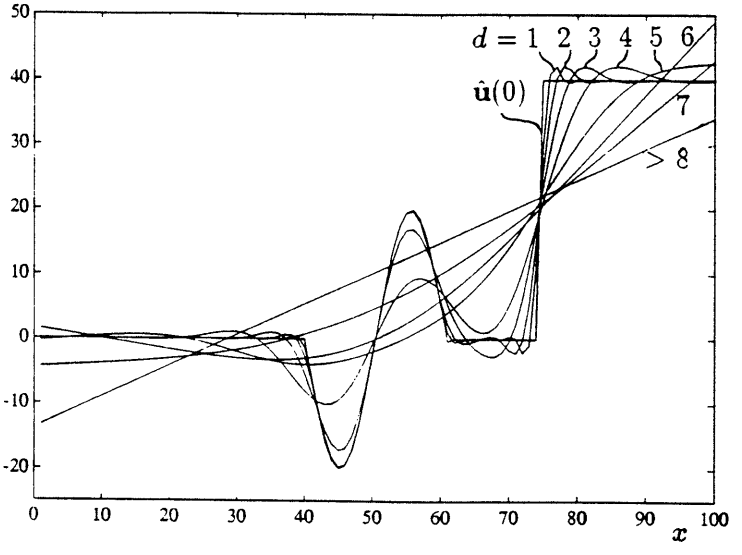

Figure 4: $\hat{\mathbf{u}}(k)$ after $k = 10^d$ times of smoothing ($p = 0.008\dot{3}$)

In order to derive restrictions for a minimum rate of updating every region, the dynamics of $\mathbf{S}^k\mathbf{u}$ going to its regression line $\mathbf{H}\,\mathbf{u}$ should be considered.

Eq. (24) is the spectral form of the matrix $\mathbf{S}^k$. The eigenvectors are orthonormal to each other. Demmler and Reinisch 1975 [59] (or [13] p.463, 465) proved that for

$i \geq 3$ the number of sign changes in the i-th eigenvector of a cubic spline smoother is $i-1$. The eigenvectors look like polynomials of degree $i-1$.

Any input vector $\hat{\mathbf{u}}$ is split up into its orthogonal components by projecting $\hat{\mathbf{u}}$ on the eigenvectors $\mathbf{v}_i$ ([58] p.148). This projection is very similar to the classic Fourier-decomposition. Instead of decomposing $\hat{\mathbf{u}}$ in its $\sin(i\omega x)$-components, $\hat{\mathbf{u}}$ is decomposed by (24) into its polynomial-like components $\mathbf{v}_i$. The components with higher "frequencies" (high number of sign changes) have small coefficients λ_i. Thus, during k times of smoothing (λ_i^k) these components go faster to zero than lower frequent components which have eigenvalues λ_i closer to one. This is exactly what is supposed to happen when the input vector $\hat{\mathbf{u}}$ is smoothed (see again Fig.4).

The last remaining non-linear component is the one having the largest eigenvalue less than one (i.e. λ_3) as coefficient. If the input vector $\mathbf{u}_{par}$ looks like a parabola and is equal to the eigenvector $\mathbf{v}_3$ then the projections onto all other eigenvectors are zero. In this case $\hat{\mathbf{u}}(k)$ can be written with (24) as

$$\hat{\mathbf{u}}(k) = \mathbf{S}^k \mathbf{u}_{par} = 0 + \lambda_3^k \mathbf{v}_3 \mathbf{v}_3^T \mathbf{u}_{par} + 0 + \ldots 0 = \lambda_3^k \mathbf{u}_{par} = e^{-\frac{k}{\ln(\lambda_3)}} \mathbf{u}_{par} = \mathbf{u}_{par} e^{-\frac{t}{\tau}} \quad (25)$$

with $t = kT$, where T is the sampling period, and with $\tau = -T/\ln(\lambda_3)$. For $\lambda_3 \approx 1$, the time constant can be estimated by

$$\tau \approx \frac{T}{1-\lambda_3}. \quad (26)$$

For low dimensions ($n < 10$) of $\mathbf{S}$ the eigenvalue λ_3 can simply be computed with the help of a standard program package. For higher dimensions an estimation formula for $\lambda_3(\mathbf{S})$ would be helpful.

$\mathbf{S}$ depends on the smoothing parameter p, the number of points n, and the sampling interval Δx. The same dependencies hold for the eigenvalues. The eigenvalues are characteristic values for the smoothing behavior (24) and therefore they should not depend on n and Δx, but only on the smoothing parameter. Some authors use pn in eq. (14) instead of p or divide the first term by n which is equivalent [3, 52, 60]. In order to make the minimization truly dimensionless

$$p = p_{sm}\, n\, (b-a)^3 \quad (27)$$

must be used in eq. (14), where p_{sm} is the independent smoothing parameter and $b - a = n\,\Delta x$ (7),(8). p_{sm} is usually a very small number (e.g. 10^{-10}).

Estimation 1 *If $p_{sm} \leq 10^{-4}$ and $n \geq 4$ then $1-\lambda_3$ can be estimated with an error less than 10% by*

$$1 - \lambda_3(\mathbf{S}) \approx 751\, p_{sm}. \quad (28)$$

With (26) τ can be expressed as

$$\tau \approx \frac{T}{751\, p_{sm}}. \quad (29)$$

In (28),(29) p_{sm} can be replaced by p by inverting (27).

Example: $p = 8.\dot{3} \cdot 10^{-3}, n = 100, \Delta x = 1, b - a = 100$ was used in Fig.3 and Fig.4. With (27) $p_{sm} = 8.\dot{3} \cdot 10^{-11}$ follows (28) $1 - \lambda_3 \approx 6.26 \cdot 10^{-8}$ and (29) $\tau = 15\,980\,000\,T$. □

Silverman 1984 shows in his paper [61] that spline smoothing can be seen as variable kernel smoothing when writing (17) as

$$\hat{u}_i(k+1) = \mathbf{s}_i\, \hat{\mathbf{u}}(k+), \qquad \forall\, i \in \{1 \ldots n\}, \tag{30}$$

where the kernel $\mathbf{s}_i$ is the i-th row of $\mathbf{S}$. The kernel $\mathbf{s}_i$ is centered at i. The shape and therefore the bandwidth bw is almost constant over all i, except near the boundaries of the interval (Fig.5).

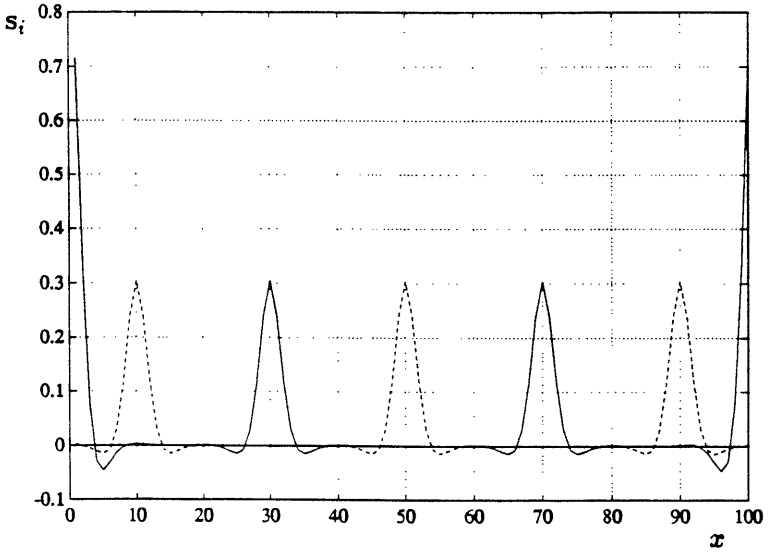

Figure 5: Kernels $\mathbf{s}_i$ for $i = 1, 10, 30, 50, 70, 90, 100$ ($n = 100, p = 1$)

The bandwidth bw of the kernels can either be estimated [61, 62] or directly measured at one of the middle rows (e.g. $\mathbf{s}_{n/2}$) of $\mathbf{S}$. The bandwidth bw is a measurement of how many points in the neighborhood of an updated point (x_j, u_j) are directly influenced by this update.

Conclusion: If t_{max} is the maximum updating time-period over all possible intervals $[x_j - bw, x_j + bw] \in \mathcal{X}$, then one should choose

$$t_{max} \ll \tau, \tag{31}$$

in order to get feasible convergence and stability results.

6.3 Infinitely many times of smoothing with fixed points

In real applications it is usually impossible to guarantee that all points will be updated equally many times. What will happen if only some of the points are updated frequently while other points are never updated? The main motivation for this paper was to find an automatically interpolating algorithm that avoids adaptation holes. Does the algorithm really converge to an interpolation of the more frequently updated points?

Let l points of $\hat{\mathbf{u}}$ with indices $i \in \mathcal{I}_{fix} = \{i_1 \dots i_l\}$ be updated very frequently, or even better, be perfectly updated. To get an idea of what is going on, we can analyze the following model: assume a continuous perfect updating process of the l points $\mathbf{u}_c = (u_{i_1} \dots u_{i_l})^T$ in such a way that we can see them as constants. Now, we smooth infinitely many times by minimizing the cost function of (15), but with the additional boundary condition $\mathbf{u}_c(k+1) = \mathbf{u}_c(k)$.

With the Lagrange multiplicands [63] written in the vector

$$\boldsymbol{\mu} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}, \qquad c_i = \begin{cases} \mu_i & \text{if } i \in \mathcal{I}_{fix} \\ 0 & \text{if } i \notin \mathcal{I}_{fix} \end{cases} \tag{32}$$

the problem can be formulated to solve

$$\hat{\mathbf{u}}(k+1) = \underset{\hat{\mathbf{u}},\boldsymbol{\mu}}{\operatorname{argmin}} \left[\|\hat{\mathbf{u}}(k) - \hat{\mathbf{u}}\|_2^2 + p\, \hat{\mathbf{u}}^T \mathbf{K} \hat{\mathbf{u}} + \boldsymbol{\mu}^T \left(\hat{\mathbf{u}}(k) - \hat{\mathbf{u}}\right) \right]. \tag{33}$$

By setting $\frac{\partial}{\partial \hat{\mathbf{u}}} = \mathbf{0}$, $\frac{\partial}{\partial \boldsymbol{\mu}} = \mathbf{0}$ and introducing a rectangular "cut"-matrix[5] of dimension $l \times n$

$$\mathbf{C} = \{c_{ji}\}, \quad c_{ji} = \begin{cases} 1 \text{ for } i = i_j \\ 0 \quad \text{else} \end{cases} \qquad \forall\, i \in \mathbf{N} | i \le n, \ \forall\, j \in \mathbf{N} | j \le l, \tag{34}$$

cutting away all non-constant $\hat{u}_i$ (e.g.: $\mathbf{u}_c = \mathbf{C}\,\hat{\mathbf{u}}$), the solution of (33) is [6]

$$\hat{\mathbf{u}}(k+1) = \mathbf{S}_{\mathcal{I}_{fix}}\, \hat{\mathbf{u}}(k), \tag{35}$$

where

$$\mathbf{S}_{\mathcal{I}_{fix}} = \mathbf{S} \left[\mathbf{I} + \mathbf{C}^T (\mathbf{C}\,\mathbf{S}\,\mathbf{C}^T)^{-1} \mathbf{C}\,(\mathbf{I} - \mathbf{S}) \right]. \tag{36}$$

[5] E.g. if $n = 13$ and $l = 3$ fixed points at x_3, x_7, and x_{12}, then $\mathcal{I}_{fix} = \{i_1 = 3, i_2 = 7, i_3 = 12\}$ and $\mathbf{C} = \begin{pmatrix} 0&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0 \end{pmatrix}$.

[6] The computation is given in Appendix A.

Theorem 2 *If* $\hat{\mathbf{u}}(0)$ *is smoothed infinitely many times by* $\hat{\mathbf{u}}(k+1) = \mathbf{S}_{\mathcal{I}_{fix}} \hat{\mathbf{u}}(k)$, *where* $\mathbf{S}_{\mathcal{I}_{fix}}$ *is defined in eq. (36),* $l \geq 2$, *then* $\lim_{k\to\infty} \hat{\mathbf{u}}(k) = \mathbf{u}_{unc}$ *is the univariate natural*[7] *cubic (unc) spline interpolation of the fixed points* $\{(x_{i_j}, u_{i_j})\}_{j=1}^{l}$. *This means that*

$$\mathbf{S}^{\infty}_{\mathcal{I}_{fix}} = \mathbf{S}_{unc}, \tag{37}$$

where $\mathbf{S}_{unc}$ *is the* $(n \times n)$*-hat matrix for the univariate natural cubic spline interpolation of the* l *fixed points.*

Proof: Every multiplication by $\mathbf{S}_{\mathcal{I}_{fix}}$ minimizes[8]

$$N = \underbrace{\sum_{i=1}^{n} (\hat{u}_i(k) - \hat{u}_i(k+1))^2}_{N_1} + \underbrace{p \int_{-\infty}^{\infty} (\hat{u}''(k+1))^2 dx}_{N_2}, \tag{38}$$

over the subspace $\mathcal{S}_{fix}$ of the Sobolev Space W_2 of functions $\hat{u}(k+1, x)$ with $\hat{u}'$ absolute continuous and $\hat{u}'' \in \mathrm{L}_2$ and

$$\hat{u}_i(k+1) = \hat{u}_i(k), \qquad \forall\, i \in \mathcal{I}_{fix}. \tag{39}$$

Denote the minimum of N in (38)

$$N(k+1) = N_1(k+1) + N_2(k+1) \tag{40}$$

and $N_{2opt} = \min_{u \in \mathcal{S}_{fix}} N_2$. Then it is known that this minimum of N_2 is only reached by the univariate natural cubic spline interpolation u_{unc} which is unique (e.g.: Wahba 1989 [57]).

a) First we show that if the series $\hat{\mathbf{u}}(k)$ is convergent, it converges to $\mathbf{u}_{unc}$. Assume there exists an $\mathbf{u}^* = \mathbf{S}_{\mathcal{I}_{fix}} \mathbf{u}^* \neq \mathbf{u}_{unc}$. Set $\mathbf{u}^*$ for $\hat{\mathbf{u}}(k)$ and $\hat{\mathbf{u}}(k+1)$ in (38), then it remains to minimize $N = 0 + N_2$ which is N_{2opt}. N_{2opt} is only reached by $\hat{\mathbf{u}}(k+1) = \mathbf{u}_{unc}$ which is a contradiction.

b) Assume $N_2(k) = N_{2opt}$, in other words $\hat{u}(k) = u_{unc}$, then no further optimization is possible. Choosing $\hat{u}(k+1) = \hat{u}(k)$ makes $N_1(k+1)$ minimal $(=0)$ and $N_2(K+1)$ minimal $(=N_{2opt})$. Thus, starting with $\hat{u}(k) = u_{unc}$, all $\hat{u}(k+j)$ will be equal to u_{unc}.

c) Assume $N_2(k) \neq N_{2opt}$. Then $N_2(k) > N_{2opt}$. A trivial trial of minimizing N would be to choose $\tilde{u}(k+1) = \hat{u}(k)$ resulting in $\widetilde{N}(k+1) = 0 + \widetilde{N}_2(k+1)$ with $\widetilde{N}_2(k+1) = N_2(k)$. This cannot be the minimizing solution as a result of part a) of this proof: $\hat{\mathbf{u}}(k+1) = \hat{\mathbf{u}}(k)$ is only possible if $N_2(k) = N_{2opt}$ which is a contradiction.

---

[7] The term *natural* comes from the *natural* boundary conditions: instead of integrating the square of the second derivative from x_1 to x_n it is integrated from $-\infty$ to $+\infty$ which makes the function $u(x)$ a straight line outside the interval $[x_1, x_n]$ according to the minimization of this term (see eq. (38)). The same result is achieved by the boundary conditions $\hat{u}''(x_1) = \hat{u}''(x_n) = \hat{u}'''(x_1) = \hat{u}'''(x_n) = 0$.

[8] Note that $\hat{u}$ denotes a continuous function of x, $\hat{u}_i$ are the values of $\hat{u}(x)$ at $x = x_i$ composing the vector $\hat{\mathbf{u}} = (\hat{u}_1, \hat{u}_2 \ldots \hat{u}_n)^T$.

Since we know the existence of a solution, there must be a solution with $N(k+1) < \widetilde{N}(k+1) = N_2(k)$. With $N_1(k+1)$ always ≥ 0 and (40) follows

$$N_2(k+1) < N_2(k). \tag{41}$$

Thus, if $k \to \infty$, $N_2(k) \to N_{2opt}$ [63, p.297] and because of uniqueness $\hat{u}(k) \to u_{unc}$. □

In case of $l = 0$, Theorem 1 provides the solution (regression line). It is mentioned without proof that for $l = 1$ the solution converges to the regression line through the fixed point.

Conclusion: The purpose of this section was to show that if some points are never updated but only smoothed through global spline smoothing then the non-updated points will converge to the cubic spline interpolation of the frequently updated points (see also Fig. 9).

7 Local Spline Smoothing

The learning algorithm using the global spline smoothing works pretty well as long as updates are provided which are evenly distributed over the whole range of x_i-values. It does not matter if some of the points are never updated, they converge to the spline interpolation of the frequently updated points (section 6.3). The worst case for the global smoothing algorithm is to update only one point x_j for a very long period. This case can easily happen in practice (e.g.: consider the learning of some car-engine characteristics during a cruise control ride on a flat straight freeway). Due to the global smoothing, in this case all the knowledge about $\mathbf{u}$ would be averaged resulting in a straight regression line through the perfectly updated point at x_j. Methods of turning on and off the learning algorithm are not recommended. In almost every application a counter-example can be constructed where the on-off-turning of the algorithm causes problems (e.g.: consider again the car on the freeway: the temperature will change during the ride and therefore $\mathbf{u}$ will change, too; learning is required all the time).

Forgetting the older points in an averaging way can be useful in some applications. Nevertheless in most cases global smoothing is not desired and local smoothing would be appreciated.

From the literature the "k-nearest neighbor estimators" (e.g.: [54, 13, 6]) are known: every point is smoothed but only the k-nearest neighbors are used to compute each of them. Contrary to this approach, we need a smoother, called *local smoother*, which smoothes only a local neighborhood around the newly updated point but uses as many points as necessary to compute the smoothed points. The problem is to fit the smoothed neighborhood in a smooth way into the unchanged part outside the neighborhood.

7.1 Moving window

The idea of local update-spline-smoothing is to define a window $[x_{j-r}, x_{j+r}]$ around x_j, to update the point at x_j in the usual way, but to smooth only the neighboring points of x_j within the window ($2r+1$ points). In this case eq. (18) cannot be used to compute the window-smoothing matrix $\mathbf{S}_{local}$. The regular spline smoother from eq. (18) would change all points inside the window without considering the required smoothness at the boundaries of the window. The smoothness is necessary in order to guarantee the smoothness of $\hat{\mathbf{u}}(k)$ for all the time (see feature 4 in section 1). Three methods of local smoothing are presented, all providing the smoothness at the boundaries.

7.1.1 All points outside the window are fixed

One way to provide the smoothness would be to calculate $\mathbf{S}_{local}$ using eq.(36) as $\mathbf{S}_{local}(x_j) = \mathbf{S}_{\mathcal{I}_{fix}}$ with $\mathcal{I}_{fix} = \{\text{all } i \text{ from 1 to } n \text{ without } i = \{j-l\}_{l=-r}^{l=+r}\}$. In other words, all points outside the window are chosen as fixed points. The disadvantage of this approach is that $\mathbf{S}_{local}(x_j)$ is dependent on x_j and therefore not constant any more as it was in eq. (18).

7.1.2 Three fixed points on each side

As we are dealing with cubic splines, it is sufficient to consider for smoothness only the continuity of the zeroth, first, and second derivative at the boundary. Thus a better approach is to define only the next three points outside the window on both sides to be fixed points and to calculate $\mathbf{S}_{local}$ using again eq. (36).

Define

$$w = 2r + 1 \tag{42}$$

to be the number of points within the window. As shown in Fig.6, we can take the w points of the window and the six points outside the window (3 on each side) and calculate the new points $\hat{u}_i(k+1)$ within the window. We need $w + 6$ input values and have w output values (it would not make sense to compute the 6 fixed points because they remain constant anyway). Thus, the matrix $\mathbf{S}_{local}$ must only have the dimension $w \times (w + 6)$, which is much less than $n \times n$.

As the points are equally spaced it makes no difference which local part of the vector $\hat{\mathbf{u}}(k+)$ is smoothed. $\mathbf{S}_{local}$ is always the same constant matrix and not dependent on j. A problem occurs if j is less than $r + 4$ or greater than $n - r - 3$. In this case the input values of the matrix multiplication (upper oval in Fig.6) exceeds the boundaries of the vector $\hat{\mathbf{u}}(k+)$. The simplest method of solving this problem is to augment the data vector with u_1 on the left side and with u_n on the right side. Now, $\mathbf{S}_{local}$ is really a constant matrix.

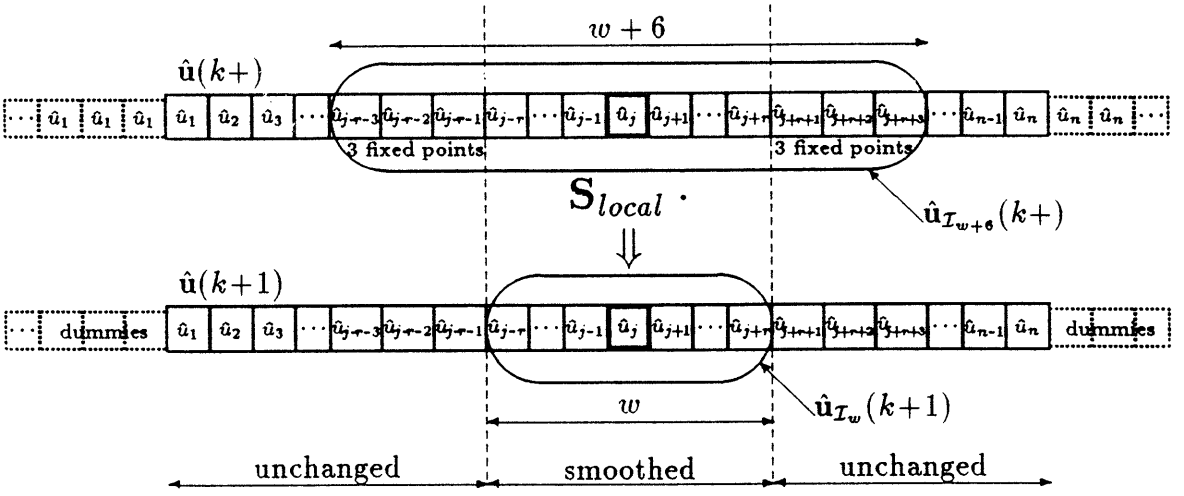

Figure 6: Illustration of eq. (47):
After $\hat{u}_j(k)$ is updated to $\hat{u}_j(k+)$ only the w points within the window are smoothed instead of smoothing the whole vector $\hat{\mathbf{u}}$. The next three points outside the window are used for the smoothing process but are considered as fixed points in order to fit smoothly the smoothed part into the unchanged part of the vector.

According to the augmentation method define

$$\text{augm}(i) = \begin{cases} 1 & \text{if } i \leq 1 \\ n & \text{if } i \geq n \\ i & \text{else.} \end{cases} \tag{43}$$

Other methods of generating data outside the vector can also be considered. Further, let us define some sets of indices: The set of n indices of $\hat{\mathbf{u}}$ is $\mathcal{I} = \{i\}_{i=1}^{n}$. The set of w indices within the window around j is

$$\mathcal{I}_w = \mathcal{I}_w(j) = \{\text{augm}(i)\}_{i=j-r}^{i=j+r}, \tag{44}$$

and the set of $w+6$ indices for the window including the 3 fixed points is

$$\mathcal{I}_{w+6} = \mathcal{I}_{w+6}(j) = \{\text{augm}(i)\}_{i=j-r-3}^{i=j+r+3}. \tag{45}$$

The set of indices outside the window is

$$\mathcal{I}_{\overline{w}} = \mathcal{I}_{\overline{w}}(j) = \mathcal{I} \text{ without } \mathcal{I}_w(j). \tag{46}$$

The vector $\hat{\mathbf{u}}(k+)$ as it was used until now, can also be written as $\hat{\mathbf{u}}_{\mathcal{I}}(k+)$ and using eq.(44) to (46), parts of the vector can be denoted as well.

Using this notation the local spline smoothing algorithm becomes

$$\left.\begin{aligned} \underbrace{\hat{\mathbf{u}}_{\mathcal{I}_w}(k+\mathrm{i})}_{\dim:\, w\times 1} &= \underbrace{\mathbf{S}_{local}}_{w\times(w+6)}\ \underbrace{\hat{\mathbf{u}}_{\mathcal{I}_{w+6}}(k+)}_{(w+6)\times 1} \\ \hat{\mathbf{u}}_{\mathcal{I}_{\overline{w}}}(k+1) &= \hat{\mathbf{u}}_{\mathcal{I}_{\overline{w}}}(k+) \end{aligned}\right\} \tag{47}$$

where $\mathbf{S}_{local}$ can be calculated from (36) and (34) as

$$\mathbf{S}_{local} = 4^{\text{th}} \text{ to } w{+}3^{\text{rd}} \text{ row of } \left[\mathbf{S}\left(\mathbf{I} + \mathbf{C}^T(\mathbf{C}\,\mathbf{S}\,\mathbf{C}^T)^{-1}\mathbf{C}\,(\mathbf{I}-\mathbf{S})\right)\right] \tag{48}$$

with

$$\mathbf{C} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & & & & & & & \\ 0 & 1 & 0 & & & & & \mathbf{0} & & \\ 0 & 0 & 1 & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\ & & \mathbf{0} & & & & & 0 & 1 & 0 \\ & & & & & & & 0 & 0 & 1 \end{pmatrix}}_{\dim:\, 6\times(w+6)} \tag{49}$$

and with S as defined in eq.(18), (16) but $\dim(\mathbf{S}) = (w{+}6)\times(w{+}6)$. Not only does eq.(47) represent a local smoother with the mentioned advantages, it also requires muss less computational effort ($w(w{+}6)$ instead of n^2) than the global smoother of eq.(18).

7.1.3 Roll-off method

The reader may think that it is much easier to simply smooth the points in the window without considering the boundary conditions and to do gradual blending afterwards, taking the unchanged values at the boundaries of the window, the regularly smoothed value in the center of the window, and a combination of both in between (a modified idea of the piecewise smoothing method by Paihua 1978 [64] and Utreras1979 [65] p.214).

Define the normed roll-off function

$$\alpha(\xi) = \xi^2(3-2\xi) \tag{50}$$

with $\alpha(0) = \alpha'(1) = 0$ representing the weight of the first point outside the window, and $\alpha(1) = 1, \alpha'(1) = 0$ representing the weight at the center of the window. Then compose the $w \times w$ weighting diagonal matrix

$$\mathbf{W} = \mathrm{diag}\left(\alpha(\tfrac{1}{r+1}), \alpha(\tfrac{2}{r+1}), \alpha(\tfrac{3}{r+1}), \cdots, \underbrace{\alpha(\tfrac{r+1}{r+1})}_{=1}, \alpha(\tfrac{r}{r+1}), \alpha(\tfrac{r-1}{r+1}), \cdots, \alpha(\tfrac{2}{r+1}), \alpha(\tfrac{1}{r+1})\right) \tag{51}$$

With $\dim \mathbf{S} = w \times w$ and $\mathbf{S}$ from (18) the smoothing process becomes

$$\left.\begin{aligned} \hat{\mathbf{u}}_{\mathcal{I}_w}(k+1) &= \mathbf{W}\,\mathbf{S}\,\hat{\mathbf{u}}_{\mathcal{I}_w}(k+) + (\mathbf{I}-\mathbf{W})\,\hat{\mathbf{u}}_{\mathcal{I}_w}(k) \\ \hat{\mathbf{u}}_{\mathcal{I}_{\overline{w}}}(k+1) &= \hat{\mathbf{u}}_{\mathcal{I}_{\overline{w}}}(k+) \end{aligned}\right\} \tag{52}$$

gradually blending the two vectors $\mathbf{S}\hat{\mathbf{u}}(k+)$ and $\hat{\mathbf{u}}(k)$ within the window. The vectors $\hat{\mathbf{u}}(k)$ and $\hat{\mathbf{u}}(k+)$ differ from each other only in one component: the updated component in the center of the window. The weight corresponding to this component is 1 as shown in (51), or 1 minus the weight is zero. Thus, the first line of (52) can also be written as

$$\begin{aligned} \hat{\mathbf{u}}_{\mathcal{I}_w}(k+1) &= \mathbf{W}\,\mathbf{S}\,\hat{\mathbf{u}}_{\mathcal{I}_w}(k+) + (\mathbf{I}-\mathbf{W})\,\hat{\mathbf{u}}_{\mathcal{I}_w}(k+) = \\ &= (\mathbf{I} + \mathbf{W}\,(\mathbf{S}-\mathbf{I}))\,\hat{\mathbf{u}}_{\mathcal{I}_w}(k+) \end{aligned}$$

or

$$\hat{\mathbf{u}}_{\mathcal{I}_w}(k+1) = \tilde{\mathbf{S}}_{local}\hat{\mathbf{u}}_{\mathcal{I}_w}(k+) \tag{53}$$

with

$$\tilde{\mathbf{S}}_{local} = \mathbf{I} + \mathbf{W}\,(\mathbf{S}-\mathbf{I}). \tag{54}$$

a) b)

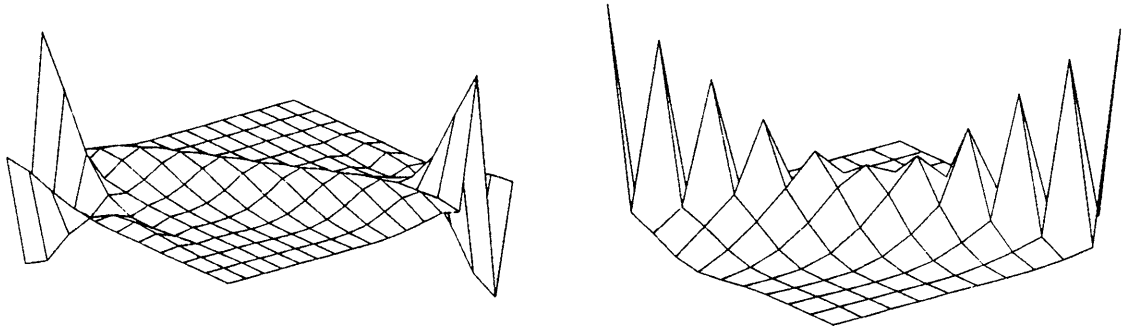

Figure 7: Smoothing matrices for $p = 1, r = 5, w = 11$
a) $\mathbf{S}_{local}$ from eq.(48), dim: 17×11,
b) $\tilde{\mathbf{S}}_{local}$ from eq.(54), dim: 11×11.

The matrix $\tilde{\mathbf{S}}_{local}$ (54) is surprisingly different from $\mathbf{S}_{local}$ in eq.(48) (see Fig.7). The two matrices represent the two different methods. The fixed point method (Fig. 7a) considers the three points outside the window to provide the boundary smoothness. The roll-off method (Fig. 7b) considers only the points inside the window. In the case of updating only one point for a very long period, the window remains at the

same place and the points within the window converge again to their regression line[9], causing a discontinuity at the boundary of the window.

The motivation for this chapter was to avoid the global smoothing. Global smoothing causes a slow forgetting of all points of $\hat{\mathbf{u}}$ in an averaging way, even if the points are far away from the actually updated point. Therefore we asked for local smoothing. It is obvious that a smoothing method cannot be recommended which is able to cause discontinuities like the roll-off method.

7.2 Example

For testing purposes both smoothing methods, the global one based on eq. (17) and the local one based on eq. (47), are applied to learn the same goal vector $\mathbf{u}$ which is representing the input-output-map (eq. (9)). The vector $\mathbf{u}$ is chosen to be the same as in Fig. 3. The selections of which point is updated next are performed randomly. 10 000 updates are done. The same parameters $p = 1$ and $c_{incr} = 1$ are used for both methods. According to eq. (47) the only difference is that local smoothing is done exclusively within a window of $w = 11$ points ($r = 5$).

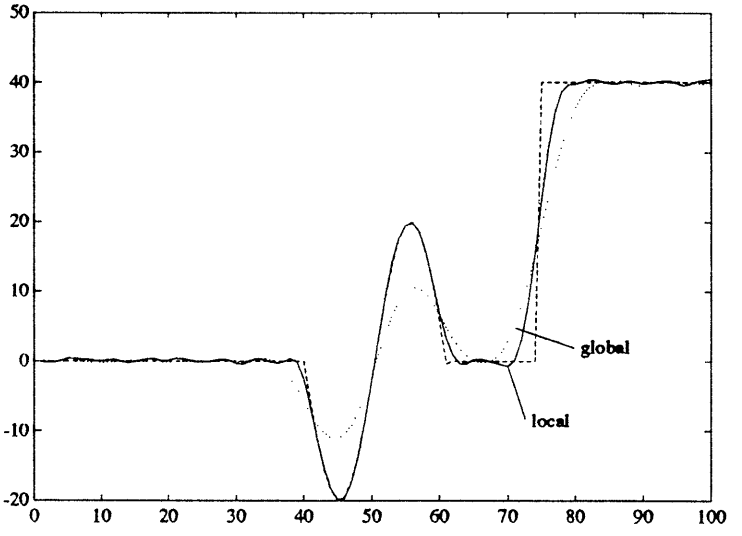

Figure 8: Different learning ability of global and local update-smoothing algorithm.
– – – – goal vector $\mathbf{u}$,
..... $\hat{\mathbf{u}}(10\,000)$ using global update-spline-smoothing,
_____ $\hat{\mathbf{u}}(10\,000)$ using local update-spline-smoothing.

[9]It is easy to show with (54) that $\lambda_i(\widetilde{\mathbf{S}}_{local}) = 1 + w_{ii}(\lambda_i(\mathbf{S}) - 1)$ and both matrices have the same eigenvectors. With $w_{ii} \neq 0$ is $\lambda_i(\widetilde{\mathbf{S}}_{local}) = 1$ iff $\lambda_i(\mathbf{S}) = 1$. As shown for eq.(24) this makes $\widetilde{\mathbf{S}}^{\infty}_{local} = \mathbf{H}$, where all matrices are of dimension $w \times w$.

In this case (see Fig. 8), the behaviors of the global and the local methods are quite different. Both converge to a stable vector $\hat{\mathbf{u}}$. The local one approximates $\mathbf{u}$ very well, except the unsmooth parts (as desired). The global one is improperly dimensioned. The smoothing operation is much stronger than the updating operation and therefore $\hat{\mathbf{u}}$ never reaches its goal $\mathbf{u}$. This fact is notable as both methods are using the same parameters. The reason for this behavior is that in the global case all points are smoothed repeatedly with every update. This makes the smoothing effect per point approximately $\frac{n}{w}$ times higher than in the local smoothing case.

Fig. 9 shows another test example. This time the global smoothing parameter is chosen $p_{global} = 0.01$, the local again $p_{local} = 0.1$. Both methods behave similarly, as long as all regions are updated (the case that updates take place only in some regions of the vector $\hat{\mathbf{u}}$ (e.g. between $x = 60$ and 90) is not shown in the figure). To make the test of Fig. 9 more interesting, only every fifth point (marked with "o" in Fig. 9) is updated and all other points are never updated. Additionally to the result of the local and global update-smoothing method, the natural cubic spline interpolation of the updated points is plotted.

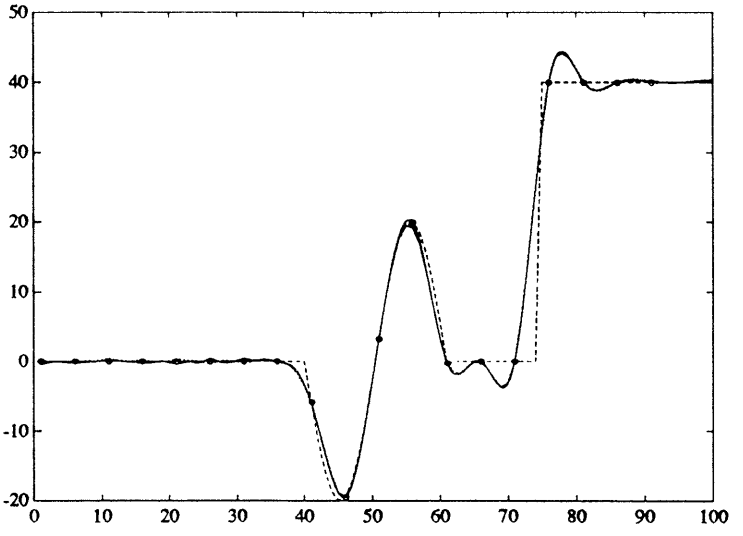

Figure 9: Learning of the input-output-map in case of sparse updated points.

- o updated points,
- – – – goal vector $\mathbf{u}$,
- ..... $\hat{\mathbf{u}}(10\,000)$ using global update-spline-smoothing,
- _____ $\hat{\mathbf{u}}(10\,000)$ using local update-spline-smoothing,
- _____ univariate natural cubic spline interpolation to updated points o.

We can see that all the non-updated points converge to the natural cubic spline interpolation of the updated points (see Theorem 2).[10] It is obvious that the goal

[10]Remember that the term *convergence*, as defined in eq. (20), tolerates a deviation of approximately $\pm c_{incr}$ which is chosen to be 1 in these examples.

vector $\mathbf{u}$ cannot be approximated very accurately by $\hat{\mathbf{u}}$ in regions where the spline interpolation to the well updated points is very different from $\mathbf{u}$. Either you have to provide the algorithm with enough updated points or you face the fact that all other points are interpolated.

The stable vector $\hat{\mathbf{u}}(\infty)$ is independent of the initial estimate $\hat{\mathbf{u}}(0)$ in all these examples.

8 Realization

Experiments have shown that the proposed update-smoothing algorithm is in general robust against bad computational resolution. Using $\frac{1}{256} \cdot \text{round}(256\, \mathbf{S}_{local})$ instead of $\mathbf{S}_{local}$ and $\text{round}(\hat{\mathbf{u}})$ instead of $\hat{\mathbf{u}}$, a similar performance is provided as with full precision.

If the computational resolution is very low, it can happen that one point is updated, but after the smoothing the ordinate of this point is exactly the same as it was before updating this point. In other words some undesired stable conditions can occur. For the same reason the interpolation ability of the algorithm can be less than expected. Thus, it is recommended to choose c_{incr} larger by an order of magnitude than the computational resolution. A factor of 2 can be sufficient in uncritical applications.

The rest of the realization is simple. Eq. (48) shows how to calculate[11] $\mathbf{S}_{local}$ which is a rectangular matrix of low dimension. Therefore, the computational effort for the matrix multiplication (smoothing operation) is low.

9 Conclusion

The issue is to track a slowly time-varying, nonlinear, but smooth characteristic curve during the life time of the system. The simplest method of learning this parameter free characteristic curve is to store in an input-output-map for every possible input value the corresponding output value. In this case, no computations are necessary to answer a query concerning one of these input-output pairs. The corresponding value is obtained simply by looking it up in the map.

Of course, this comfort of very cheap queries has to be paid by some effort for the updating operation. It is known that the unknown input-output-map is a smooth input-output relationship with the option to be very slowly time-varying. All new information about this relationship has to be used, but the smoothness of the relationship must not get lost.

In order to avoid "adaptation holes" or any discontinuities, cubic spline smoothing is done after every update (so called *update smoothing*). The paper discussed two

[11] **MATLAB is a very helpful tool for computing this matrix.**

different smoothing methods: the global and the local spline smoothing.

Global smoothing should not be used in applications where it cannot be guaranteed that all input regions are updated within some time period. Otherwise the smoothing task averages the already learned relationship, converging to the straight regression line of the data points. The time constant of this forgetting process was estimated.

The better way is to use local smoothing, smoothing only a local neighborhood of the updated point. Even if we stay for thousands of updates at the same point, the relationship outside of the neighborhood window is not destroyed.

It was possible to show that the update-spline-smoothing algorithm has the interpolation property. The never or seldom updated points converge to the univariate natural cubic spline interpolation of the well updated points. This requires the trivial restriction that at least as many points have to be frequently updated that the cubic spline interpolation of these points is sufficiently accurate.

There are still many open problems in this field. More concrete design rules would be appreciated for the one-dimensional problem, as treated in this paper. The learning of hysteresis functions and some application oriented details have to be solved.

Furthermore, the multi-dimensional update-smoothing should be solved. Different smoothing methods, the neural network approach (Tolle 1989 [66], Poggio 1990 [1], Atkeson 1989 [7], Livstone 1991 [67], Lippmann 1987 [68]), and other learning methods (e.g. Horowitz 1989 [69]) should be considered.

Appendix A

Solving eq. (33):

$$
\begin{aligned}
\frac{\partial}{\partial \hat{\mathbf{u}}} = \mathbf{0} &= -2(\hat{\mathbf{u}}(k) - \hat{\mathbf{u}}) + 2p\mathbf{K}\hat{\mathbf{u}} - \boldsymbol{\mu} && (a)\\
\frac{\partial}{\partial \boldsymbol{\mu}} = \mathbf{0} &= \mathbf{C}\hat{\mathbf{u}}(k) - \mathbf{C}\hat{\mathbf{u}} && (b)\\
\text{with (a):}\quad \hat{\mathbf{u}}(k+1) = \hat{\mathbf{u}} &= (\mathbf{I} + p\mathbf{K})^{-1}(\hat{\mathbf{u}}(k) + \tfrac{\boldsymbol{\mu}}{2})\\
\text{with (18):}\quad \hat{\mathbf{u}} &= \mathbf{S}\hat{\mathbf{u}}(k) + \mathbf{S}\tfrac{\boldsymbol{\mu}}{2} && (c)\\
\mathbf{C}\cdot \Longrightarrow\quad \mathbf{C}\hat{\mathbf{u}} &= \mathbf{CS}\hat{\mathbf{u}}(k) + \mathbf{CS}\tfrac{\boldsymbol{\mu}}{2}\\
\text{with (b):}\quad \mathbf{C}\hat{\mathbf{u}}(k) &= \mathbf{CS}\hat{\mathbf{u}}(k) + \mathbf{CS}\tfrac{\boldsymbol{\mu}}{2}\\
\text{or:}\quad \mathbf{CS}\tfrac{\boldsymbol{\mu}}{2} &= \mathbf{C}(\mathbf{I} - \mathbf{S})\hat{\mathbf{u}}(k)\\
\text{with } \boldsymbol{\mu} = \mathbf{C}^T\mathbf{C}\boldsymbol{\mu}:\quad \mathbf{CSC}^T\mathbf{C}\tfrac{\boldsymbol{\mu}}{2} &= \mathbf{C}(\mathbf{I} - \mathbf{S})\hat{\mathbf{u}}(k)\\
(\mathbf{CSC}^T)^{-1}\cdot \Longrightarrow\quad \mathbf{C}\tfrac{\boldsymbol{\mu}}{2} &= (\mathbf{CSC}^T)^{-1}\mathbf{C}(\mathbf{I} - \mathbf{S})\hat{\mathbf{u}}(k)\\
\mathbf{C}^T\cdot \Longrightarrow\quad \tfrac{\boldsymbol{\mu}}{2} &= \mathbf{C}^T(\mathbf{CSC}^T)^{-1}\mathbf{C}(\mathbf{I} - \mathbf{S})\hat{\mathbf{u}}(k)\\
\text{with (c):}\quad \hat{\mathbf{u}}(k+1) &= \underbrace{\mathbf{S}\left[\mathbf{I} + \mathbf{C}^T(\mathbf{CSC}^T)^{-1}\mathbf{C}(\mathbf{I} - \mathbf{S})\right]}_{\mathbf{S}_{\mathcal{I}_{fix}}}\hat{\mathbf{u}}(k) && \text{Eq.(35),(36)}
\end{aligned}
$$

Acknowledgements

The author wishes to thank Prof. Sanjoy K. Mitter, Prof. Gilbert Strang, Prof. Christopher Atkeson, Prof. Walter Olbricht, Prof. John N. Tsitsiklis (all M.I.T.), Prof. Grace Wahba (University of Wisconsin at Madison), Prof. Jerome H. Friedman (Stanford University), Dr. Andreas Buja (Bellcore), and Dorothea Czedik-Eysenberg (Boston University) for their helpful discussions.

References

[1] T. Poggio and F.Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, February 1990.

[2] E. Grosse. A catalog of algorithms for approximation. *netlib@research.att.com*, pages 1–27, 1989.

[3] G. Wahba. Smoothing noisy data with spline functions. *Numerische Mathematik*, 24:383–393, 1975.

[4] J. H. Friedman. Multivariate adaptive regression splines. Technical report 102, Department of Statistics, Stanford University, November 1988.

[5] E. J. Wegman and I. W. Wright. Splines in statistics. *Journal of the American Statistical Association*, 78(382):351–365, 1983.

[6] Ch. G. Atkeson. Memory-based approches to approximating continous functions. note, M.I.T., the Artificial Intelligence Laboratory, 1990.

[7] Ch. G. Atkeson. Learning arm kinematics and dynamics. *Annual Review of Neuroscience*, 12:157–183, 1989.

[8] J. H. Friedman and W.Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 78(376):817–823, 1981.

[9] P. J. Green. Semi-parametric generalized linear models. In *Lecture Notes in Statistics 32, Generalized Linear Models*, pages 44–55, Berlin Heidelberg New York, 1985. Springer-Verlag.

[10] Ph. J. Davis. *Interpolation and Approximation.* Blaisdell Publishing Company, New York Toronto London, 1963.

[11] L. L. Schumaker. Fitting surfaces to scattered data. In G. G. Lorentz, C. K. Chui, and L. L. Schumaker, editors, *Approximation Theory III*, pages 203–268, New York, 1976. Academic Press.

[12] R. Franke. Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, 38(157):181–200, 1982.

[13] A. Buja, T. Hastie, and R. Tibshirani. Linear smoothers and additive models. *The Annals of Statistics*, 17(2):453–555, 1989.

[14] J. Moody. Fast learning in multi-resolutional hierarchies. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 29–39. Morgan Kaufmann Publisher, San Mateo, CA, 1989.

[15] M. J. D. Powell. Radial basis functions for multivariable interpolation: a review. In J. C. Mason and M. G. Cox, editors, *Algorithms for Approximation*, pages 143–167, Oxford, 1987. Clarendon Press.

[16] R. E. Bellman. *Adaptive Control Processes.* Princeton University Press, Princeton, New Jersey, 1961.

[17] Ch. G. Atkeson and J. McIntyre. Robot trajectory learning through practice. In *IEEE International Conference on Robotics and Automation*, pages 1737–1742, 1986.

[18] Ch. G. Atkeson. Using locally weighted regression for robot learning. In *IEEE International Conference on Robotics and Automation*, Sacramento, CA, April 1991.

[19] Ch. G. Atkeson. Memory-based learning control. In *American Control Conference (ACC 91)*, pages 2131–2136, Boston, 1991. IEEE.

[20] D.-Y. Yeung and G. A. Gekey. Using a context-sensitive learning network for robot arm control. In *IEEE International Conference on Robotics and Automation*, pages 1441–1447, May 1989.

[21] M. Heiss. *Minimization of Dead Time in Electronic Diesel Contol Systems (in German).* PhD thesis, University of Technology, Vienna, Austria, August 1989.

[22] Ch. Augesky and M. Heiss. Einrichtung zum Steuern und Regeln einer Brennkraftmaschine eines Fahrzeuges. German Patent DE 3822582, July 1988.

[23] M. Heiss, Ch. Augesky, A. Seibt, and W. Bittinger. Einrichtung zum Steuern und Regeln einer Dieselbrennkraftmaschine. German Patent Application DE 3906083, February 1989.

[24] U. Kiencke and C. T. Cao. Regelverfahren in der elektronischen Motorsteuerung I, II. *Automobilindustrie*, pages 629–636 and 135–144, 6/1987 and 2/1988.

[25] H. Yamaguchi. Motor vehicle with line pressure control for automatic transmission. European Patent Application EP 0398057, 1990.

[26] R. Bitzer and F. Piwonka. Verfahren und Stellregler zur adaptiven Regelung eines reibungsbehafteten Antriebs. German Patent Application DE 3731983, 1989.

[27] M. Silberbauer. Digitaler Mengenservo. Technical Report 55, Voest Alpine Automotive, Vienna, Austria, March 1990.

[28] K. Abe. Learning control systems for controlling an automotive engine. US-Patent 4733357, 1988.

[29] K. Abe. Kraftstoff-Luftverhältnis-Überwachungssystem für eine Kraftfahrzeugmaschine. German Patent Application DE 4001494, 1990.

[30] N.Tomisawa. Learning and control apparatus for electronically controlled internal combustion engine. US-Patent 4763627, 1988.

[31] P. Schmidt and M.Schmitt. Verfahren und Einrichtung zur Steuerung/Regelung von Betriebsgrößen einer Brennkraftmaschine. German Patent Application DE 3603137, 1988.

[32] M. H. Furuyama. System zum Erfassen anormaler Betriebszustände eines Verbrennungsmotors. German Patent Application DE 3819016, 1988.

[33] M. H. Furuyama. Steuerungssystem für das Luft-Kraftstoff-Verhältnis für einen KFZ-Motor. German Patent Application DE 3928585, 1990.

[34] S. Miyama. Verfahren zur Steuerung des Zündzeitpunktes einer Brennkraftmaschine. German Patent Application DE 4018447, 1990.

[35] H. Ohishi. Verfahren und Vorrichtung zum Regeln des Luft-Kraftstoff-Verhältnisses des einer Brennkraftmaschine für ein Fahrzeug zugeführten Luft-Kraftstoff-Gemisches. German Patent DE 3726867, 1990.

[36] T. Hori, T. Atago, and M. Nagano. Method of controlling air-fuel ratio for use in internal combustion engine and apparatus of controlling the same. European Patent Application EP 0358062, 1989.

[37] K. Kameta, K. Morita, T. Kikuchi, and Y. Tanabe. Method and apparatus for controlling fuel supply to an internal combustion engine. European Patent Application EP 0339585, 1989.

[38] M. Holmes. Einstell-Regelsystem für einen Verbrennungsmotor und Verfahren zum Regeln eines Verbrennungsmotors. German Patent DE 3703496, 1989.

[39] E. Linder, M. Klenk, and W. Moder. Steuer-/Regelsystem für instationären Betrieb einer Brennkraftmaschine. German Patent Application DE 3731983, 1989.

[40] S. Nakinawa and T. Tomisawa. Control system for internal combustion engine with improved control characteristics at transition of engine conditions. European Patent Application 0314081, 1989.

[41] K. Walter. Kraftstoffregeleinrichtung für Brennkraftmaschinen mit Regelschaltung zum Erfassen der Eichung und Verfahren zum Betrieb von Brennkraftmaschinen. German Patent Application DE 2829958, 1979.

[42] S. Sasaki, Y. Mouri, and N. Sugiura. Engine control device. European Patent 0155663, 1989.

[43] M. Klenk. Lernendes Regelungsverfahren für eine Brennkraftmaschine und Vorrichtung hierfür. German Patent Application DE 3811262, 1989.

[44] M. Klenk. Lernendes Regelungsverfahren für eine Brennkraftmaschine und Vorrichtung hierfür. German Patent Application DE 3811263, 1989.

[45] M. Heiss. Adaption von Kennlinien in Echtzeit. *Elektrotechnik und Informationstechnik e & i*, 106(10):398–402, 1989.

[46] I. J. Schoenberg. Spline functions and the problem of graduation. *Proc. Nat. Acad. Sci. U.S.A.*, 52:947–950, 1964.

[47] Ch. H. Reinisch. Smoothing by spline functions. *Numerische Mathematik*, 10:177–183, 1967.

[48] C. De Boor. *A Practical Guide to Splines.* Springer, New York, 1978.

[49] G. Wahba. *Spline Models for Observational Data.* CBMS-NSF # 59. Society for Industrial and Applied Mathmatics, Philadelphia, 1990.

[50] P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerische Mathematik*, 31:377–403, 1979.

[51] L. L. Schumaker. *Spline Functions: Basic Theory.* Wiley, New York, 1981.

[52] R. L. Eubank. The hat matrix for smoothing splines. *Statistics & Probability Letters*, 2:9–14, 1984.

[53] R. L. Eubank. Diagnostics for smoothing splines. *J. R. Statist. Soc. B*, 47(2):332–341, 1985.

[54] R. L. Eubank. *Spline Smoothing and Nonparametric Regression.* Marcel Dekker, New York and Basel, 1988.

[55] B. W. Silverman. Some aspects of the spline smoothing approach to nonparametric regression curve fitting. *J. R. Statist. Soc. B*, 47(1):1–52, 1985.

[56] E. T. Whittaker. On a new method of graduation. *Proc. Edinburgh Math. Soc.*, 41:63–75, 1923.

[57] G. Wahba. Spline functions. In *Encyclopedia of Statistical Science, Supplement Volume*, pages 148–159. John Wiley & Sons, New York, 1989.

[58] G. Strang. *Linear Algebra and its Applications.* Harcourt Brace Jovanovich, San Diego, 3rd edition, 1988.

[59] A. Demmler and C. Reinisch. Oscillation matrices with spline smoothing. *Numerische Mathematik*, 24:375–382, 1975.

[60] D. D. Cox. Asymptotics for M-type smoothing splines. *The Annals of Statistics*, 11(2):530–551, 1983.

[61] B. W. Silverman. Spline smoothing: The equivalent variable kernel method. *The Annals of Statistics*, 12(3):898–916, 1984.

[62] W. Härdle. *Applied Nonparametric Regression.* Cambridge University Press, Cambridge, New York, New Rochelle, Melbourne, Sydney, 1990.

[63] I. N. Bronstein and K. A. Semendjajew. *Taschenbuch der Mathematik.* Teubner, Leipzig, 20th edition, 1981.

[64] L. Paihua. *Quelques Methodes Numeriques pour les Fonctions Spline a une et deux Variables (in French).* PhD thesis, Grenoble, May 1978.

[65] F. Utreras. Cross-validation techniques for smoothing spline functions in one or two dimensions. In *Lecture Notes in Mathematics 757, Smoothing Techniques for Curve Estimation*, pages 196–232, Berlin Heidelberg New York, 1979. Springer-Verlag.

[66] H. Tolle, J. Militzer, and E. Ersü. Zur Leistungsfähigkeit lokal verallgemeinernder assoziativer Speicher und ihren Einsatzmöglichkeiten in lernenden Regelungen. *Messen Steuern Regeln msr*, 32(3):98–105, 1989. See also 9^{th} IFAC World Congr. pp. 1039-1044.

[67] M. M. Livstone, J. A. Farrell, and W. L. Baker. A computationally efficient algorithm for training recurrent networks. *IEEE Transactions on Neural Networks (submitted)*, 1991.

[68] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[69] R. Horowitz, W.-W. Kao, M. Boals, and N. Sadegh. Digital implementation of repetitive controllers for robotic manipulators. In *IEEE International Conference on Robotics and Automation*, volume III, pages 1497–1503, May 1989.

[70] J. G. Hayes and J. Halliday. The least-squares fitting of cubic spline surfaces to general data sets. *J. Inst. Maths Applics*, 14:89–103, 1974.

[71] W. Th. Miller III, F. H. Glanz, and L. G. Kraft III. Application of a general algorithm to the control of robotic manipulators. *The International Journal of Robotics Research*, 6(2):84–98, 1987.