Czech Technical University in Prague
Faculty of Electrical Engineering

BACHELOR'S THESIS

Steady State of Fuzzy Dynamical System

Prague, 2015

Author: Šimon Pavlík

Prohlášení

Prohlašuji, že jsem svou diplomovou (bakalářskou) práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ____________ ________________
podpis

Acknowledgement

I would like to thank my supervisor Doc. Ing. Petr Hušek, Ph.D. for his guidance. I am very grateful for the many valuable and interesting things he has taught me. I would also like to thank my family and relatives for their patience and support throughout my studies.

Abstract

The main objective of my thesis is to design a fuzzy system using Recursive Least Squares Algorithm (RLS), which will be applied to approximate a grey-box dynamic non-linear model, provided we know beforehand, that the model to be identified is monotonic. Stand-alone algorithm will be proposed, which will enforce monotonicity of the system after each step of RLS. Finally, two different monotonic functions will be identified in order to demonstrate the algorithm's effect on the system's behavior.

Abstrakt

Hlavním cílem mé bakalářské práce je navrhnout fuzzy systém s využitím rekurzivní metody nejmenších čtverců (RLS), která bude použita k aproximaci grey-box dynamického nelineárního modelu, o němž předem víme, že je monotónní. Bude navržen samostatný algoritmus, jenž zaručí monotonicitu systému po každém kroku RLS. Nakonec se pokusíme identifikovat dvě různé monotonní funkce, za účelem demonstrace vlivu algoritmu na chování systému.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Šimon Pavlík**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Ustálený stav fuzzy dynamického systému**

Pokyny pro vypracování:

1. Seznamte se s principy fuzzy logiky a popisem dynamického systému pomocí fuzzy modelu
2. Pro fuzzy dynamický systém 1. řádu nalezněte jeho převodní charakteristiku.
3. Stanovte pravidla, za kterých bude převodní charakteristika fuzzy dynamického systému 1. řádu monotónní.

Seznam odborné literatury:

[1] F. Hoeppner et al.: Fuzzy Cluster Analysis, Wiley, 2000
[2] L. X. Wang: A course in fuzzy systems and control, Prentice Hall, 1997
[3] Z. Vlček: Analysis of Autoregressive Fuzzy Systems, FUZZ-IEEE 2004, Budapest, pp. 1233-1238

Vedoucí: Doc.Ing. Petr Hušek, Ph.D.

Platnost zadání: do konce zimního semestru 2014/2015

[redacted]

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.

[redacted]

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 27. 9. 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Thanks to their excellent approximation properties fuzzy systems are widely used in the area of non-linear black box system identification. By measuring the input-output data pairs of the system we can accurately determine its behavior without knowing the inner structure of the system.

In various situations we are provided with additional knowledge of certain basic properties of the identified system. Many physical systems for instance are known to be monotonic. A higher input value produces a higher output value. Incorporating prior information of a different nature into our model is called grey modeling.

Quality of the black box model is oftentimes limited by the scarcity of measured data, which furthermore tend to be corrupted by noise in measurements from sensors. Poor data leads to a less accurate model which may lack the desired properties.

If we manage to design a fuzzy mapping based on an inherently monotonic system without taking advantage of this prior information, we may find certain areas of the system breaching monotonicity due to errors during the identification. Behavior of the system in these areas would then produce results which contradict our expectation. For example, in a hydraulic model an increase in flow rate of the water supply could cause the water level to go down. The controller based on such wrong model would then produce opposite reaction, which could be fatal.

In the following chapters we will form a specific fuzzy system structure. We are going to use a Recursive Least Square Algorithm enhanced by an algorithm to ensure monotonicity in each training step. The Monotonicity Checking Algorithm will allow for identification of models with up to 2 inputs. We are going to carry out a few experiments in order to test the algorithm.

Chapter 2

Fuzzy System Composition

The main component of this work is the fuzzy system, which is used in place of a system that we want to identify.

2.1 IF-THEN Rules

We are going to focus on a two-dimensional (2 input) version of the fuzzy system, where fuzzy rule base is constructed for $M = N_1 \times N_2$ IF-THEN rules in the following form:

$$Ru^{i_1 i_2} : \; IF \; x_1 \; is \; A_1^{i_1} \; and \; x_2 \; is \; A_2^{i_2}, \; THEN \; y \; is \; B^{i_1 i_2} \tag{2.1}$$

where $i_1 = 1, 2, ...N_1, i_2 = 1, 2, ..., N_2,$ A represented by the membership functions is a normal fuzzy set in the input space $U \subset R^2$ and the center of the fuzzy set $B^{i_1 i_2}$ denoted by $\bar{y}^{i_1 i_2}$ belongs to the output space $V \subset R$. $\bar{y}$ is a point at which a given membership function achieves its maximum value.

2.2 Fuzzy Mapping

Our 2 input systems will be based on the two-dimensional fuzzy mapping form

$$f(x) = \frac{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \bar{y}^{i_1 i_2} (\mu_{A_1^{i_1}}(x_1) \mu_{A_2^{i_2}}(x_2))}{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} (\mu_{A_1^{i_1}}(x_1) \mu_{A_2^{i_2}}(x_2))} \tag{2.2}$$

consisting of a fuzzy rule base, product inference engine, singleton fuzzifier and center average defuzzifier with centers denoted as $\bar{y}$ and membership functions denoted as μ.

All the above concepts are well explained in (Wang, L. X., 1997).

The generalized version of the n-input fuzzy system with N fuzzy sets contains a total of N^n rules with each dimension increasing the number of rules exponentially.

We need to uniformly cover the whole set $U = [\alpha_1, \beta_1] \times [\alpha_2, \beta_2]$ by membership functions, in such a way that the fuzzy sets $A_i^1, ...A_i^{N_i}$ are complete. It means that at every $x \in U$ there exists i_1 and i_2 such that $\mu_{A_1^{i_1}}(x_1)\mu_{A_2^{i_2}}(x_2)) \neq 0$. The denominator of the fuzzy system will hence always be nonzero.

2.3 Membership Functions

All fuzzy systems in the following chapters are going to use *Triangular Membership Functions*, which come from a family of Pseudo-Trapezoid of membership functions. Fig. 2.1 shows examples of pseudo-trapezoid functions (Wang, L. X., 1997).

Definition 2.1 (Pseudo-Trapezoid Membership Function): Let $[a, d] \subset R$. The *pseudo-trapezoid membership function* of fuzzy set A is a continuous function in R given by

$$\mu_A(x; a, b, c, d, H) = \begin{cases} I(x), x \in [a, b) \\ H, x \in [b, c] \\ D(x), x \in (c, d] \\ 0, x \in R - (a, d) \end{cases} \tag{2.3}$$

where $a \leq b \leq c \leq d, a < d, 0 < H \leq 1, 0 \leq I(x) \leq 1$ is a non-decreasing function in $[a, b)$ and $0 \leq D(x) \leq 1$ is a non-increasing function in $(c, d]$. When the fuzzy set A is normal (that is, H = 1), its membership function is simply written as $\mu_A(x; a, b, c, d)$. ▶

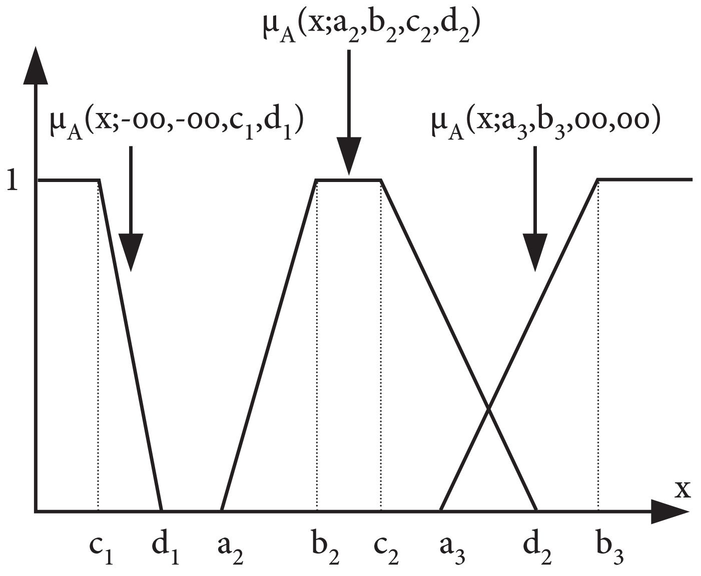


Figure 2.1: Examples of pseudo-trapezoid membership functions

Triangular Membership Function is one of the most commonly used membership functions in the field of fuzzy identification. It is the case of 2.1, where $b = c$ and

$$I(x) = \frac{x-a}{b-a}\,,\ D(x) = \frac{x-d}{c-d}\,, \tag{2.4}$$

denoted as $\mu_A(x; a, b, d)$. If we choose $a = \infty,\ b = c = \bar{x},\ d = \infty$ and

$$I(x) = D(x) = exp(-(\frac{x-\bar{x}}{\sigma})^2), \tag{2.5}$$

we obtain another commonly used membership function called *Gaussian Membership Function*. There are certain benefits of using Gaussian Membership functions to design a fuzzy system. It is continuously differentiable and therefore suitable for all fuzzy system where we require the use of derivation. Moreover, its computation is very simple.

We are now going to cover the whole space with fuzzy sets consisting of triangular membership functions each beginning at the vertex e of the preceding one in a fashion depicted in Fig. 2.2 (Wang, L. X., 1997). The first/last membership function has its beginning/ending at its vertex leaving its second half out of the fuzzy set.

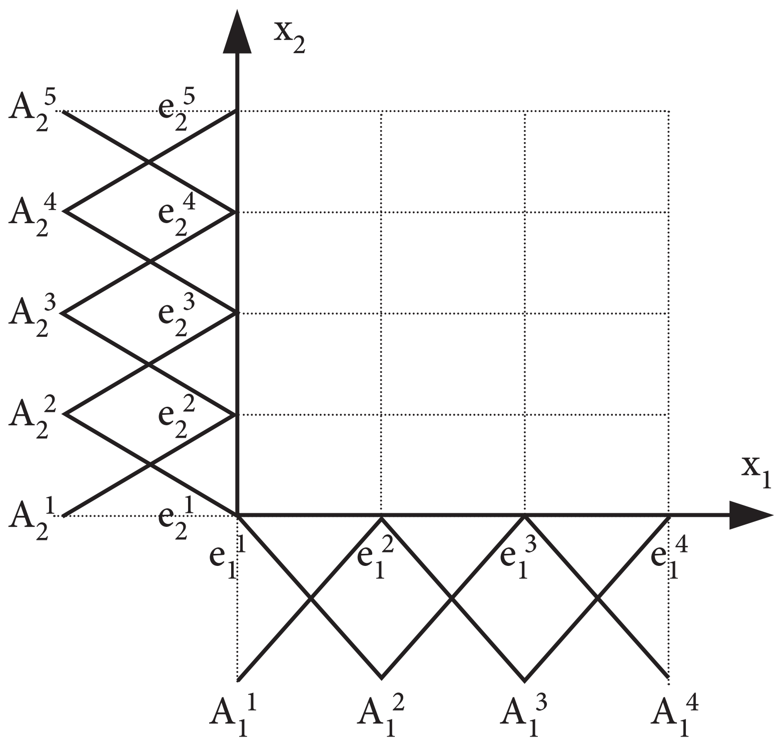


Figure 2.2: Layout of the fuzzy sets

In this particular 2-dimensional configuration with Triangular Membership Functions we have a total of 4 active fuzzy sets at any given moment. Each input is active in two fuzzy sets provided that both inputs are belong to their domain U. We are going to take advantage of this property during the monotonicity checking phase.

In contrast each fuzzy set with Gaussian Membership Function covers the whole domain, therefore it gets activated every time there is a valid incoming signal.

Another interesting attribute of the above mentioned configuration with Triangular Membership Functions shown in Fig. 2.2 is that the denominator always sums to one, further simplifying the fuzzy system into

$$f(x) = \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \bar{y}^{i_1 i_2} (\mu_{A_1^{i_1}}(x_1) \mu_{A_2^{i_2}}(x_2)). \tag{2.6}$$

Chapter 3

Recursive Least Squares

With the structure of the fuzzy system fully specified, our next step is to determine its parameters according to the input-output pairs. With the overall layout of the fuzzy sets and all membership functions fixed, the parameters of our system which are free to change are the values of centers $\bar{y}$. We are now going to proceed towards optimization of all centers on a training set of data.

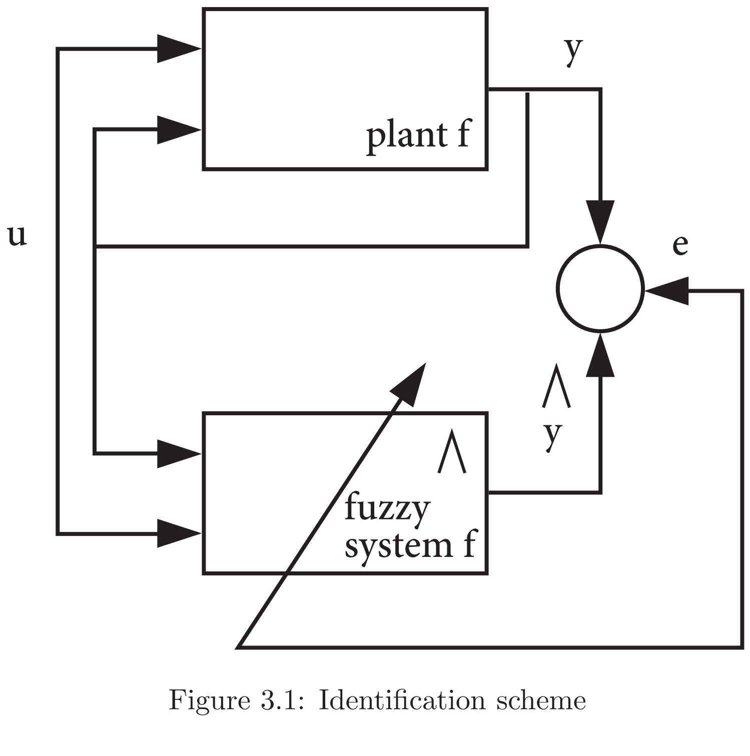


Figure 3.1: Identification scheme

3.1 Formulation of Recursive Least Squares Algorithm

3.1.1 Minimization Problem

Criterion J_p in the form

$$J_p = \sum_{j=1}^{p} [f(x^j) - y^j)]^2 \tag{3.1}$$

minimizes the sum of all squares of errors between the fuzzy system $f(x)$ and the real system y by choosing the appropriate centers $\bar{y}$ for all the input-output pairs and up to the training sample p. The identification scheme in Fig. 3.1 (Wang, L. X., 1997) shows the process of step by step training. Recursive nature of the algorithm is achieved by training one step at a time, which in effect means that the current choice of optimal parameters is a function of a single previous step. An important requirement of the selected method is that all parameters of the optimized system must be linear.

3.1.2 Fuzzy System Formula Modification

Before using the fuzzy system for two inputs designed in the previous chapter in 2.2 and 2.6, we are going to slightly modify the fuzzy mapping

$$f(x) = \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \bar{y}^{i_1 i_2} (\mu_{A_1^{i_1}}(x_1) \mu_{A_2^{i_2}}(x_2)) \tag{3.2}$$

to fit the optimization problem.

We need to collect all the centers, which will be optimized into a single vector Θ with with length $N_1 \times N_2$.

$$\Theta = (\bar{y}^{11}, .., \bar{y}^{N_1 1}, \bar{y}^{12}, .., \bar{y}^{N_1 2}, .., \bar{y}^{1 N_2}, .., \bar{y}^{N_1 N_2})^T \tag{3.3}$$

and rewrite the fuzzy system (3.2) as

$$f(x) = b^T(x)\Theta \tag{3.4}$$

where

$$b(x) = (b^{11}(x), .., b^{N_1 1}(x), b^{12}(x), .., b^{N_1 2}(x), .., b^{1 N_2}(x), .., b^{N_1 N_2}(x))^T \tag{3.5}$$

$$b^{i_1 i_2}(x) = \mu_{A_1^{i_1}}(x_1) \mu_{A_2^{i_2}}(x_2) \tag{3.6}$$

3.1.3 RLS Algorithm

For each consecutive step $p = 1, 2, ...,$ of our training data series we are going to update parameters Θ in a following way:

$$\begin{aligned} K(p) &= P(p-1)b(x_0^p)[b^T(x_0^p)P(p-1)b(x_0^p)+1]^{-1} && (3.7)\\ update(p) &= K(p)[y_0^p - b^T(x_0^p)\Theta(p-1)] && (3.8)\\ \Theta(p) &= \Theta(p-1) + update(p) && (3.9)\\ P(p) &= P(p-1) - P(p-1)b(x_0^p)\cdot \\ &\quad \cdot[b^T(x_0^p)P(p-1)b(x_0^p)+1]^{-1}b^T(x_0^p)P(p-1) && (3.10) \end{aligned}$$

The Recursive Least Squares Algorithm is derived in (Wang, L. X., 1997).

3.1.4 Initialization of Parameters

There are two parameters which determine the rate of convergence towards the optimum and both need to be initialized carefully.

Initial parameter $\Theta(0)$ should be chosen either based on certain knowledge we have about the identified system, or if we are identifying a black box system, we can simply leave all centers initialized to zero.

The second parameter initialization

$$P(0) = \sigma I \tag{3.11}$$

will be chosen in accordance with the first one. In general σ represents the confidence we put into our $\Theta(0)$ choosing. Higher values of σ are going to cause higher initial magnitudes of training updates and vice versa. The update may be exaggerated if the initial $\Theta(0)$ differs only slightly from the reality. If we choose the σ too low, convergence during the training and the final precision of the trained system could be poor.

$P(n)$ is proportional to the covariance matrix of the parameters Θ. If our knowledge of these parameters at $p = 0$ is very vague, a very high covariance matrix of the parameters is to be expected, and thus we must assign a high value to σ (Tabus,I, 2012).

3.2 Other Optimization Techniques

3.2.1 Least Squares

Apart from RLS there are various other optimization approaches at hand. Related exact method which has wide-spread applications is the classical Least Squares Method (LS). Unlike RLS, LS finds optimal parameters in a single step. LS can often produce more accurate results. However, there is a need to compute a matrix inverse, an operation which brings many limitations and should generally be avoided. It requires a lot of computational power and causes a cumulation of error due to a loss of information during its computation.

RLS on the other hand consists of many trivial computations which have much lower hardware requirements, therefore it is more cost efficient and suitable for real-time applications such as data and signal processing and control systems.

3.2.2 Gradient Descent Training

Another alternative to RLS is the Gradient Descent Training algorithm. It attempts to reach the optimum by choosing the steepest descent in every step. However due to its non-convex nature, it can get stuck in a local minimum vastly different from the optimal solution which lies in the global minimum.

3.2.3 Computational complexity of RLS and LS

The computational complexity of RLS compared to LS is extensively reduced mostly thanks to the absence of matrix inversion. The number of algebraic operations and required memory locations is reduced from $O(r^3)$ to $O(r^2)$ per each iteration (i.e. per new data point) as a function of data amount r. Moreover, even more efficient algorithms (known as "fast algorithms") have been developed, which further reduce the computational complexity to $O(r)$ (Zhu, Y.M.; Li, X.R., 2007).

Chapter 4

Monotonicity Checking Algorithm

Main purpose of monotonicity checking algorithm is to ensure after each training phase of RLS, that the identified system meets conditions of monotonicity over all trained fuzzy centers. Area over which monotonicity has been violated needs to be corrected. It is important to guarantee monotonicity in each step. The idea is to limit the extent of corrections to minimum, since each correction largely modifies the outcome of RLS. Thus the next step proceeds towards optimization by quadratic programming, which chooses optimal corrections to minimize intervention.

4.1 Conditions of Monotonicity

Definition 4.1 (Non-Decreasing Mapping): A mapping with ordered sets of x_1^i, x_2^j is non-decreasing if $\forall(i,j)\ x_1^i \leq x_1^{i+1}$, $x_2^j \leq x_2^{j+1}$:

$$f(x_1^i, x_2^j) \leq f(x_1^{i+1}, x_2^j) \tag{4.1}$$

$$f(x_1^i, x_2^j) \leq f(x_1^i, x_2^{j+1}) \tag{4.2}$$

▶

Before we look at the monotonicity checking algorithm in detail we first need to define conditions under which the fuzzy mapping 2.2 is monotonic based on 4.1. Given our chosen type of membership functions and configuration of centers it is necessary and sufficient that all neighboring pairs of centers meet the following conditions:

Proposition 4.1 (Non-Decreasing Fuzzy Mapping): *System is non-decreasing if for* *each individual center $\bar{y}^{i_1 i_2}$ of the fuzzy system 2.2*

$$\bar{y}^{i_1 i_2} \leq \bar{y}^{i_1+1 i_2} \tag{4.3}$$

$$\bar{y}^{i_1 i_2} \leq \bar{y}^{i_1 i_2+1} \tag{4.4}$$

Conditions of monotonicity are derived in (Lindskog, P.; Ljung, L., 1997).

4.2 Monotonic Initialization of RLS

Crucial step before the RLS training starts is to properly initialize the centers, which must remain monotonic from the beginning. Leaving them initialized to zero would negatively affect convergence towards optimal value. Most updates would be classified as monotonicity breaching.

Viable solution is to initialize the centers in $\Theta(0)$ with a plane. In most cases we know the range of the identified system's output. Assuming prior knowledge of the minimum and maximum value we can use it to interpolate the plane.

4.3 Monotonicity Checking

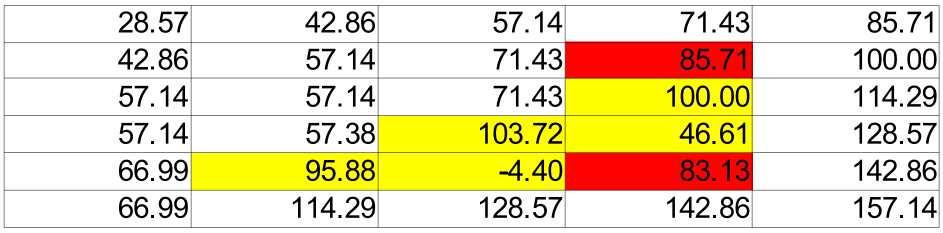
After each RLS phase we are presented with a new set of updates to each center of the fuzzy system being trained. Given a 2 dimensional system 2.2, we obtain a total of 4 rules with non-zero values of membership functions that were activated by the incoming data in each step of RLS. One would expect that it results in just 4 centers to be updated. However, because of the RLS parameter P (3.11), which depends on all previous incoming data, any center can be updated at a given step. The resulting update is therefore more complex.

There are two different strategies of ensuring monotonicity that we are going to use: One is to accept the full scale update as given by RLS. Another is to limit the scale of the update only on the area where IF conditions of the fuzzy system were activated by the recent incoming data. This way we would only allow 4 corresponding center updates, where we expect the updates to be most significant. That could effectively reduce the amount of centers where monotonicity conditions were broken and limit spreading of the non-monotonic area. On the other hand it can eventually cause an even greater modification of the original result of RLS.

Although my algorithm is primarily designed to ensure non-decreasing properties of the system, we can simply change the problem to non-increasing by providing the algorithm with the opposite value of its 2 input arguments (previous center values and updates).

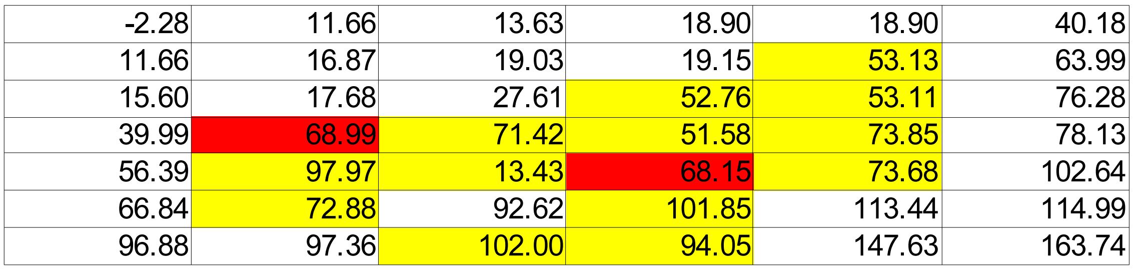
In a first step algorithm determines minimum and maximum value each center is allowed to have based on its surroundings without breaking system monotonicity. The first center has no minimum and the last has no maximum value. Violations of monotonicity are found by checking the matrix of centers against their min/max allowed values.

In certain rare configurations violations cannot be discovered simply by checking against their nearest neighborhood as we can see in the following two examples. Ordering begins in the top left corner with the smallest center. The centers depicted in yellow show the violations found so far.



28.57	42.86	57.14	71.43	85.71
42.86	57.14	71.43	85.71	100.00
57.14	57.14	71.43	100.00	114.29
57.14	57.38	103.72	46.61	128.57
66.99	95.88	-4.40	83.13	142.86
66.99	114.29	128.57	142.86	157.14

Figure 4.1: A pair of centers causing further violations.



-2.28	11.66	13.63	18.90	18.90	40.18
11.66	16.87	19.03	19.15	53.13	63.99
15.60	17.68	27.61	52.76	53.11	76.28
39.99	68.99	71.42	51.58	73.85	78.13
56.39	97.97	13.43	68.15	73.68	102.64
66.84	72.88	92.62	101.85	113.44	114.99
96.88	97.36	102.00	94.05	147.63	163.74

Figure 4.2: 2 distant centers with violations.

However, if we look further, we can see that the whole identified non-monotonic area is surrounded by centers (depicted in red) that are also mutually causing violations in relation with each other. Fig. 4.1 shows a pair of centers on the opposite sides of the identified non-monotonic area causing monotonicity violations. Fig. 4.2 shows another example of distant centers mutually causing monotonicity violations.

To ensure all the violations were thoroughly searched for we need to look around the vicinity of the non-monotonic area and check each pair of centers separately. In case of a 2 dimensional system we are comparing all the North and West pairs with East and South pairs. The area extends as we loop through until no further violations have been found and we can proceed to the next step.

4.4 Formation of Linear Inequalities

Once we have found all violations next step is to form inequalities between the two centers where corrections are required. We can distinguish the relations between a violation and its neighbor by two factors: Firstly whether the neighbor itself is also a violation and secondly, in our 2 dimensional situation, whether it is a Northern or Western neighbor (N, W) or a Southern or Eastern one (S, E). Altogether we have the following 4 types equations where c stands for a correction that is necessary in order to ensure mononotonicity of the system and v is an index of a center or a correction with monotonicity violation.

North or West neighbor without violation:

$$\bar{y}^N \leq \bar{y}^v + c^v \tag{4.5}$$

$$\bar{y}^W \leq \bar{y}^v + c^v \tag{4.6}$$

South or East neighbor without violation:

$$\bar{y}^v + c^v \leq \bar{y}^S \tag{4.7}$$

$$\bar{y}^v + c^v \leq \bar{y}^E \tag{4.8}$$

North or West neighbor with violation:

$$\bar{y}^N + c^N \leq \bar{y}^v + c^v \tag{4.9}$$

$$\bar{y}^W + c^W \leq \bar{y}^v + c^v \tag{4.10}$$

South or East neighbor with violation:

$$\bar{y}^v + c^v \leq \bar{y}^S + c^S \tag{4.11}$$

$$\bar{y}^v + c^v \leq \bar{y}^E + c^E \tag{4.12}$$

4.5 Searching for Optimal Correction

Last step is to find a set of optimal corrections for all inequalities in a way that will cause only minimal necessary modifications of the RLS outcome. The task leads to a solution by quadratic programming. We will now arrange the linear constraints above into a following form:

$$\boldsymbol{Ac} \leq \boldsymbol{b} \tag{4.13}$$

All linear coefficients of corrections c from the constraints are either +1, -1, or 0 and are collected into matrix $\boldsymbol{A}$. Vector $\boldsymbol{b}$ represents the constants of differences between the neighboring pairs of centers, e.g. $(\bar{y}^v - \bar{y}^{NW})$.

Finally, optimization criteria for n violations is written in the following form:

$$J = \sum_{i=1}^{n} c_i^2 \tag{4.14}$$

By solving a problem

$$\min_{\boldsymbol{Ac} \leq \boldsymbol{b}} (J(\boldsymbol{c})) \tag{4.15}$$

we obtain suitable set of corrections $\boldsymbol{c}$ that will return monotonicity back to our system.

Function *quadprog* used for quadratic programing in my algorithm is part of MATLAB's Optimization Toolbox.

The choice of using quadratic programming over linear programming is determined by the nature of the minimization criterion. In case of linear programming conditions could be met simply by choosing some corrections negative and others positive, which would sum up to zero total correction. However, this solution would be far from optimal. We are looking for a sum of absolute values of corrections to be minimal. That is the reason we need their squares.

4.6 Known Limitations

Precision of the algorithm is somewhat limited in the very first and the very last center of the surface, where it is the most sensitive to an extremely big or inappropriate update suggested by the RLS in the vicinity of the given center. Both centers are restricted only by a maximum value in case of the first center and a minimum value in case of the last. In order to compensate wrong update near the corner the optimal correction takes part of the update into the corner center where it has an unrestricted minimum or maximum value. A lot of additional training around the corner area is hence necessary to bring the center back to it's correct value. We can partly avoid this issue by applying the strategy mentioned at the beginning of this chapter which allows for 4 center updates. Monotonicity is then checked only on those 4 and the error cannot spread towards corners unless the update occurred there.

There is another property to be aware of during the early phases of training. Sometimes the algorithm chooses optimal correction, which leaves certain small areas flat. In this situation the result of the best correction are two or more neighboring centers with equal values.

Chapter 5

Fuzzy Approximation of a 2-variable Function

In this chapter we are going to test the fuzzy identification with enforced monotonicity by approximation of 2 monotonic functions of two variables. Output signal will contain pseudorandom noise drawn from the standard uniform distribution with a maximum magnitude 20% of its range. Training phase will be performed on 2000 steps with random input values. Fuzzy system consists of 11 membership functions on each input, which results in a total of 121 centers.

First function represents an elliptic paraboloid:

$$y_1(n) = u_1^2(n-1) + u_2^2(n-1) + noise \tag{5.1}$$

Second function is a square root:

$$y_2(n) = \sqrt{u_1(n-1)} + \sqrt{u_2(n-1)} + noise \tag{5.2}$$

5.1 Results of Enforced Monotonicity Training

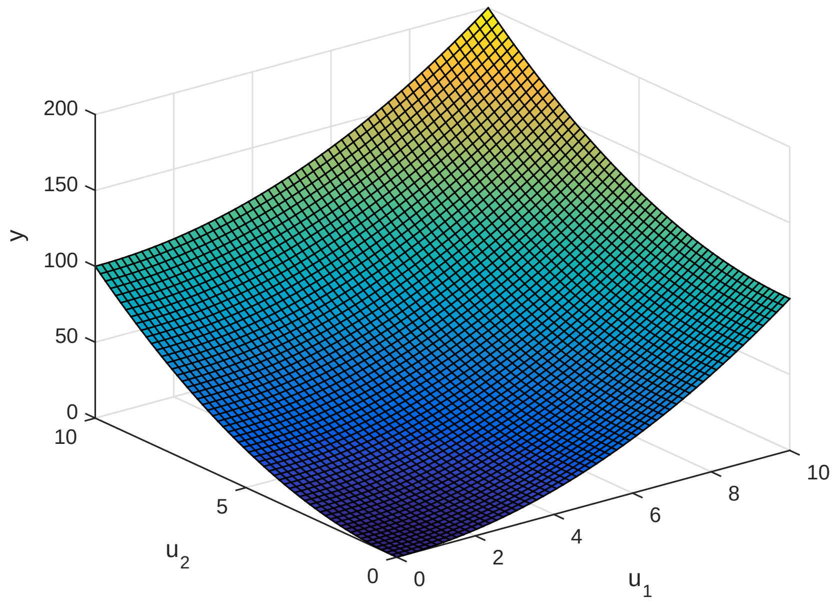


Figure 5.1: Real paraboloid surface - no noise

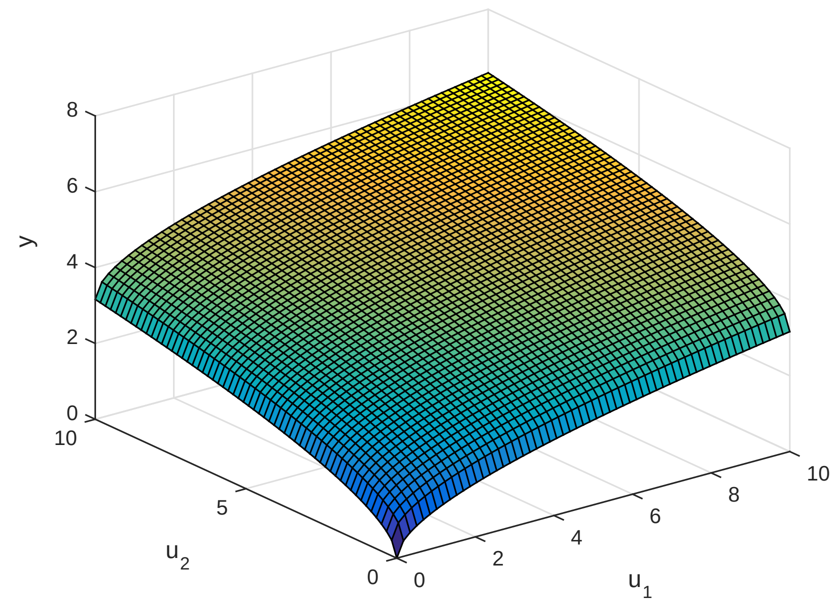


Figure 5.2: Real square root surface - no noise

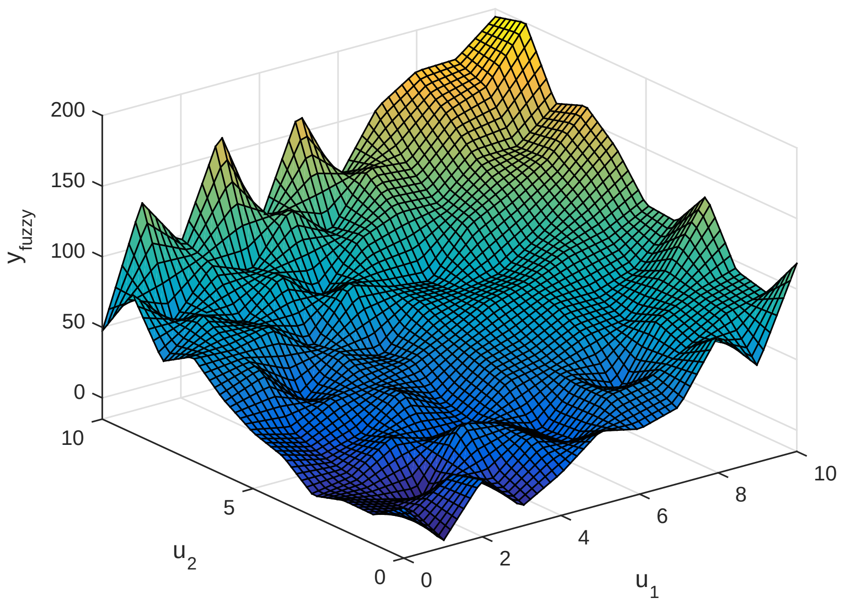


Figure 5.3: Paraboloid fuzzy identification

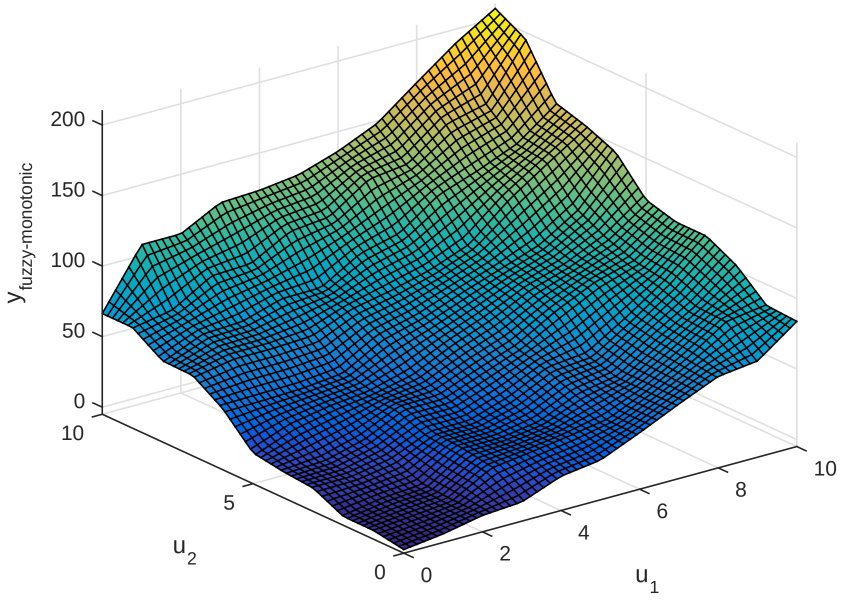


Figure 5.4: Paraboloid monotonic fuzzy identification

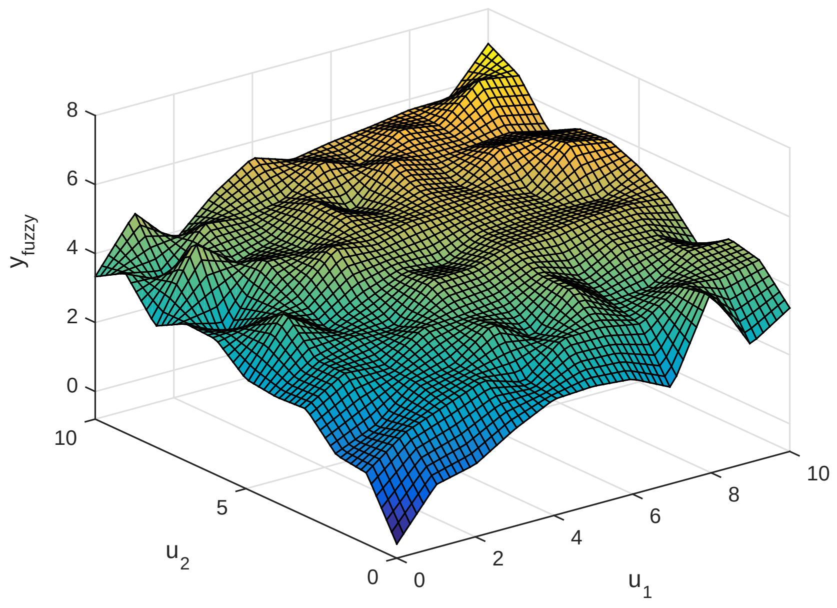


Figure 5.5: Sqrt fuzzy identification

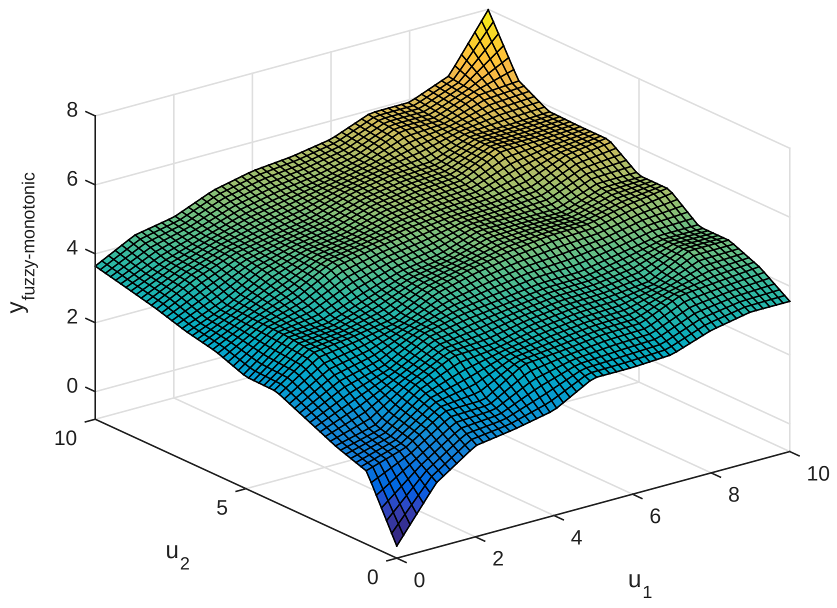


Figure 5.6: Sqrt monotonic fuzzy identification

As we can see in the charts above, high noise level in both functions causes standard fuzzy identification to greatly loose its precision. Monotonicity checking algorithm on the other hand not only successfully preserves monotonicity, it produces a surface that highly resembles the real function.

Fig. 5.6 shows an example of a small peak in the corner area of the surface. As mentioned in the previous chapter this phenomenon can sometimes occur in the area of the first and last center.

5.2 Response to a Random Input

In this experiment, we are going to measure the response of the two functions to a random input $u_1,\ u_2$:

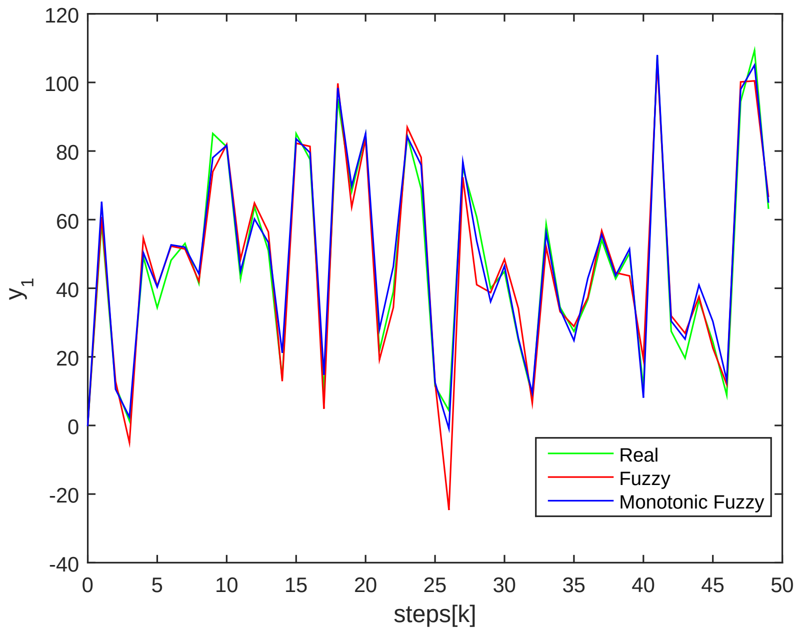


Figure 5.7: Paraboloid random input response

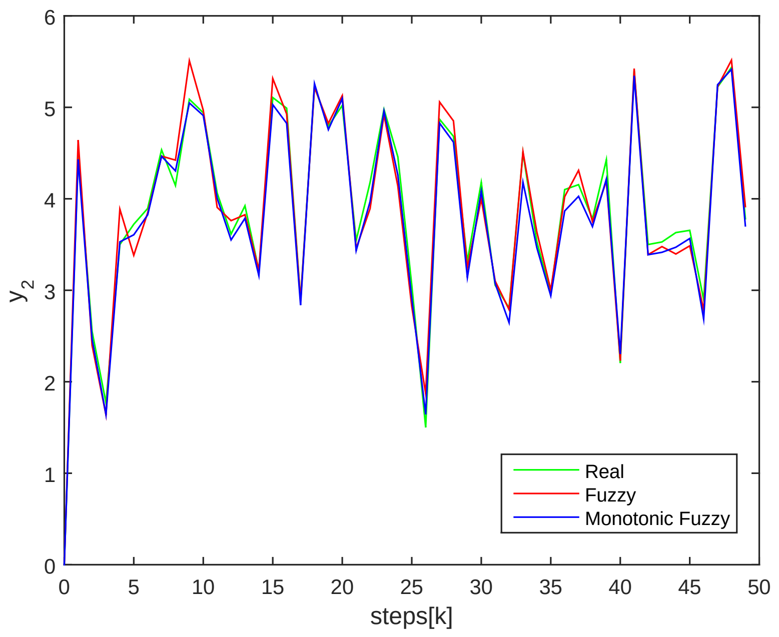


Figure 5.8: Sqrt random input response

The experiment in 5.7 and 5.8 shows that The Monotonic Fuzzy System reproduces almost perfectly whereas the Classical Fuzzy System is less smooth with various overshoots.

5.3 Accuracy with Different Levels of Noise

In order to quantitatively assess the degree of similarity between the identified fuzzy function and the real function we are going to use Root Mean Square Error (RMSE). For n acquired function values y RMSE calculation is the following:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y^i_{fuzzy} - y^i_{real})^2} \tag{5.3}$$

4 different levels of noise will be used to test the 2 strategies of training from the previous chapter.

Table 5.1: RMSE - Paraboloid

	4-center Update RMSE		Full Update RMSE	
Noise [%]	Fuzzy	Monotonic Fuzzy	Fuzzy	Monotonic Fuzzy
5	1.9249	1.6614	1.8057	1.9066
10	3.2111	2.4134	3.0428	3.2977
20	7.1775	4.8811	6.7106	5.6886
40	15.2559	7.3391	12.7160	12.5747

Table 5.2: RMSE - Square Root

	4-center Update RMSE		Full Update RMSE	
Noise [%]	Fuzzy	Monotonic Fuzzy	Fuzzy	Monotonic Fuzzy
5	0.0815	0.0732	0.0931	0.0838
10	0.1341	0.0923	0.1139	0.0993
20	0.2339	0.1621	0.2331	0.1616
40	0.5375	0.2588	0.4380	0.3960

Collected data shows only two cases where RMSE of the monotonic fuzzy system remains slightly higher than of the non-monotonic one. It is in the case of Full Update strategy under low noise conditions. With higher levels of noise, deviations of both monotonic and non-monotonic fuzzy functions remain similar.

4-center Update strategy significantly improves RMSE values of the monotonicity enforced training. Discrepancy is especially apparent under higher noise conditions. Non-monotonic RMSE is approximately twice as high. This seems intuitive as the spreading of the non-monotonic area cannot occur and the essential monotonicity corrections are less drastic.

5.4 Convergence of Centers Throughout Training

Another interesting experiment is to observe a certain center as it converges towards its desired value with each step of RLS. Different initialization parameters of RLS lead to a different starting phase. Following charts show progress of training a middle center $\bar{y}^{61}$. Overshoot within the first 300 steps can be limited by choosing the RLS parameter P smaller, which effectively puts more confidence in center initialization. The monotonic fuzzy RLS training usually blocks exaggerated updates since they usually lead to monotonicity breach. The only area susceptible to spikes, as mentioned earlier, is around the corner centers of the fuzzy system. The figures show that the two training methods more or less correlate and converge in the late phases of training.

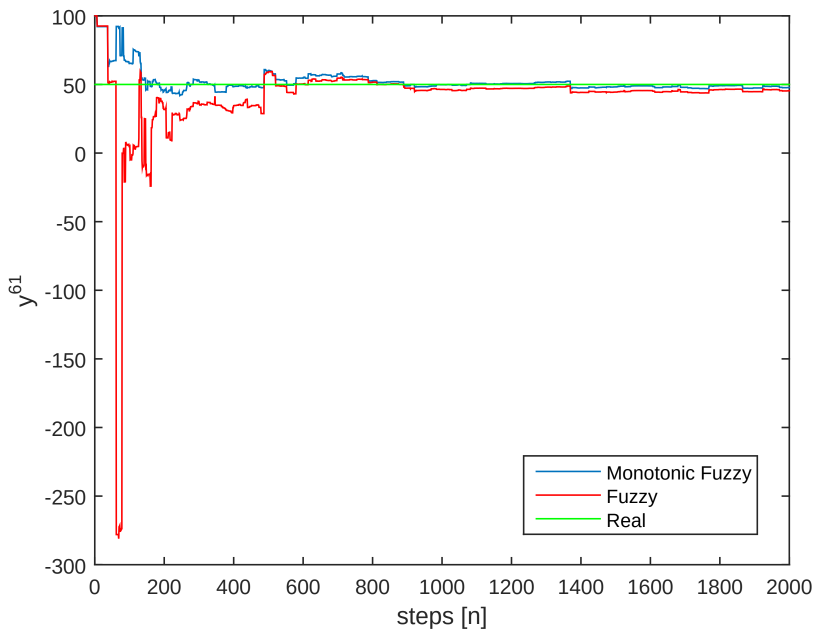


Figure 5.9: Paraboloid Full Update convergence

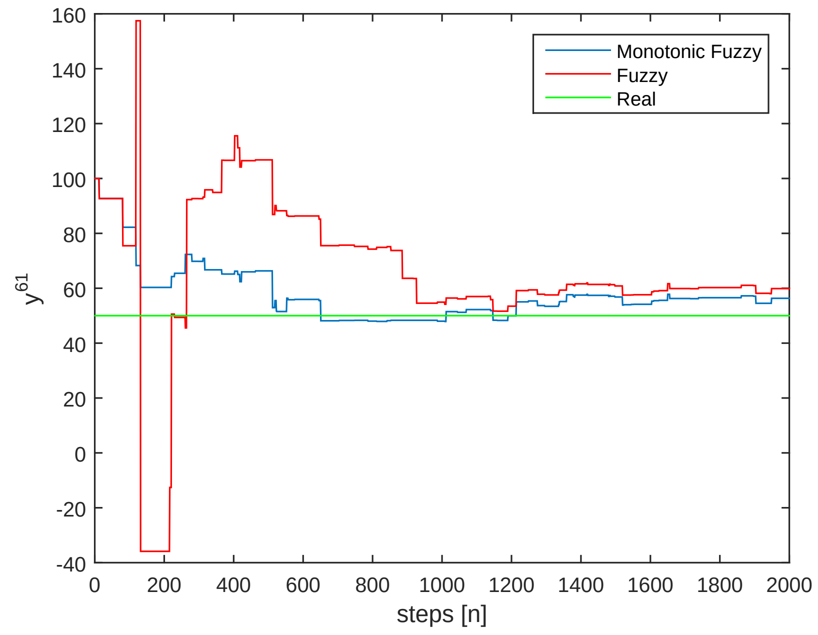


Figure 5.10: Paraboloid 4-center Update convergence

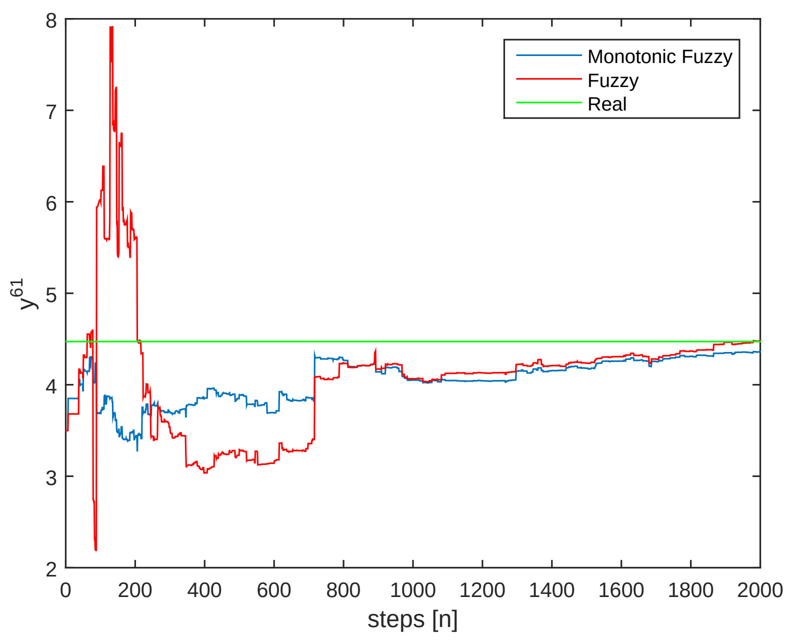


Figure 5.11: Sqrt Full Update convergence

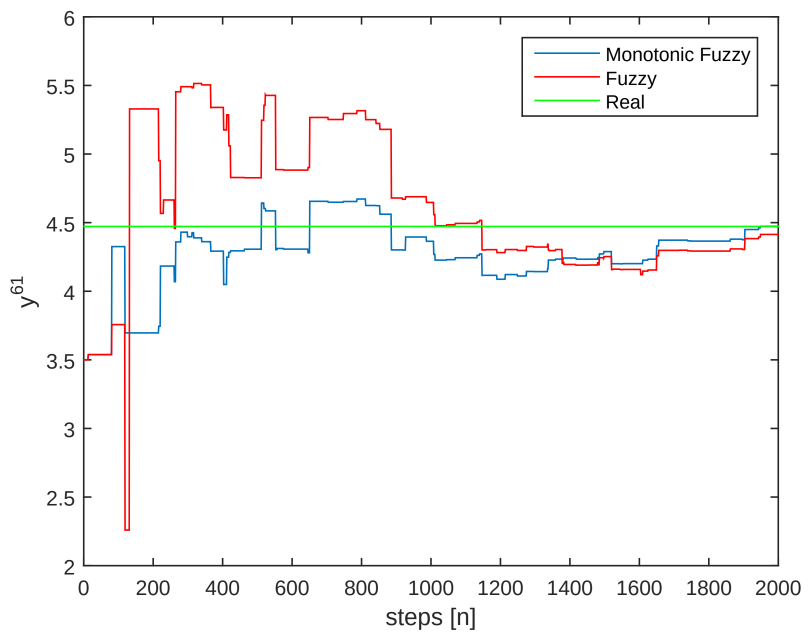


Figure 5.12: Sqrt 4-center Update convergence

5.5 Diverging Centers

In our final 5.13 we are going to demonstrate a situation where the progress of training with the monotonicity enforcing method accurately converges to the value of Real System, whereas Classical Fuzzy System diverges far above. Note that the center was initialized precisely at the real value of the system, thus resulting to a zero initial matching error of training. The chaotic behavior at the beginning is caused by noise and the large initial state of the parameter P(3.11).

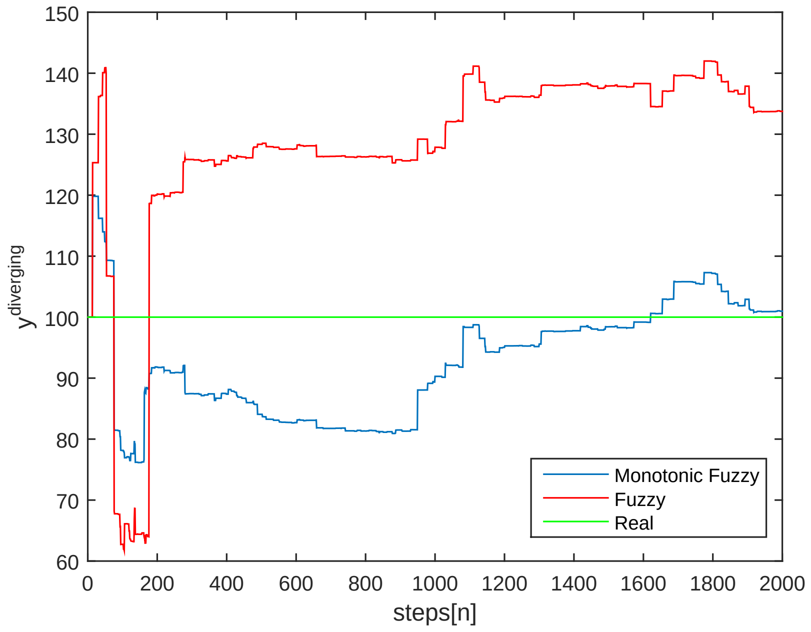


Figure 5.13: Paraboloid Full Update divergence

Chapter 6

Conclusion

We have presented a method for modeling complex non-linear systems based on fuzzy logic. We have shown a Recursive Least Squares training algorithm used for the adjustment of parameters in the identified fuzzy system. We have successfully designed and demonstrated a grey-box modeling technique, which ensures monotonicity of the identified system and it is compatible with systems of up to two inputs. Proposed algorithm used two different approaches and managed to reliably correct all monotonicity breaching issues that emerged during the identification phases with different levels of noise. Monotonicity Checking algorithm greatly improved training done by the Recursive Least Squares algorithm. Comparison was made, which lead to plausible results both quantitatively measured by the Root Mean Square Error and illustratively shown. Displayed output surface of the identified systems and progress of training show the contrast between the classical fuzzy identification and monotonicity enforced identification. Identified monotonic systems were more similar to the original models than their non-monotonic counterparts both visually and in terms of RMSE.

All computations and scripts were developed in MATLAB 2013b/2014b and they are available on the enclosed CD-ROM including many further experiments on different systems.

Bibliography

LINDSKOG, P.; LJUNG, L. (1997), 'Ensuring certain physical properties in black box models by applying fuzzy techniques', *Conference: SYSID '97* .

TABUS,I (2012), Advanced signal processing - lecture 10: Recursive least squares estimation. Tampere University of Technology http://www.cs.tut.fi/ tabus/course/ASP/LectureNew10.pdf.

WANG, L. X. (1997), *A Course in Fuzzy Systems and Control*, Prentice-Hall, Inc. ISBN 0-13-593005-7.

ZHU, Y.M.; LI, X.R. (2007), 'Recursive least squares with linear constraints', *Communications in Information and Systems* **7**(3), 287–312.