Knowledge-Based Control Systems

Robert Babuška and Jens Kober
Delft Center for Systems and Control
Version 19-07-2017

Delft University of Technology
Delft, the Netherlands

Contents

1. Introduction

This chapter gives a brief introduction to the subject of the book and presents an outline of the different chapters. Included is also information about the expected background of the reader. Finally, the Web and MATLAB support of the present material is described.

1.1. Conventional Control

Conventional control-theoretic approaches are based on mathematical models, typically using differential and difference equations. For such models, methods and procedures for the design, formal analysis and verification of control systems have been developed. These methods, however, can only be applied to a relatively narrow class of models (linear models and some specific types of nonlinear models) and definitely fall short in situations when no mathematical model of the process is available. Even if a detailed physical model can in principle be obtained, an effective rapid design methodology is desired that does not rely on the tedious physical modeling exercise. These considerations have led to the search for alternative approaches.

1.2. Intelligent Control

The term 'intelligent control' has been introduced some three decades ago to denote a control paradigm with considerably more ambitious goals than typically used in conventional control. While conventional control methods require more or less detailed knowledge about the process to be controlled, an intelligent system should be able to autonomously control complex, poorly understood processes such that some pre-specified goal can be achieved. It should also cope with unanticipated changes in the process or its environment, learn from past experience, actively acquire and organize knowledge about the surrounding world and plan its future behavior. Given these highly ambitious goals, clearly motivated by the wish to replicate the most prominent capabilities of our human brain, it is perhaps not so surprising that no truly intelligent control system has been implemented to date.

Currently, the term 'intelligent' is often used to collectively denote techniques originating from the field of artificial intelligence (AI), which are intended to replicate some of the key components of intelligence, such as reasoning, learning, etc. Among these techniques are artificial neural networks, expert systems, fuzzy logic systems, qualitative models, genetic algorithms and various combinations of these tools. While in some cases, these techniques really add some truly intelligent features to the system, in other situations they are merely used as an alternative way to represent a fixed nonlinear control law, process model or uncertainty. Although in the latter these methods do not explicitly contribute to a higher degree of machine intelligence, they are still useful. They enrich the area of control by

employing alternative representation schemes and formal methods to incorporate extra relevant information that cannot be used in the standard control-theoretic framework of differential and difference equations.

Two important tools covered in this textbook are fuzzy control systems and artificial neural networks. Fuzzy control is an example of a rule-based representation of human knowledge and the corresponding deductive processes. Artificial neural networks can realize complex learning and adaptation tasks by imitating the function of biological neural systems. The following section gives a brief introduction into these two subjects and, in addition, the basic principle of genetic algorithms is outlined.

1.3. Overview of Techniques

Fuzzy logic systems describe relations by means of if–then rules, such as 'if heating valve is open then temperature is high.' The ambiguity (uncertainty) in the definition of the linguistic terms (e.g., *high temperature*) is represented by using *fuzzy sets*, which are sets with overlapping boundaries, see Figure 1.1. In the fuzzy set framework, a particular domain element can simultaneously belong to several sets (with different degrees of membership). For instance, $t = 20°\mathrm{C}$ belongs to the set of *High* temperatures with membership 0.4 and to the set of *Medium* temperatures with membership 0.2. This gradual transition from membership to non-membership facilitates a smooth outcome of the reasoning (deduction) with fuzzy if–then rules; in fact a kind of interpolation.

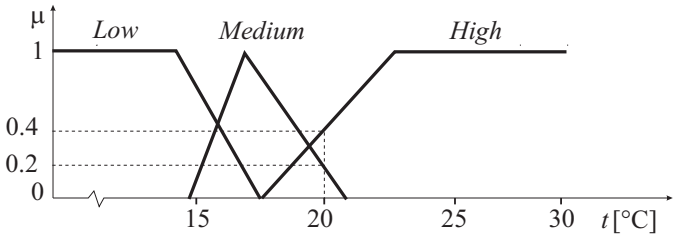


Figure 1.1.: Partitioning of the temperature domain into three fuzzy sets.

Fuzzy logic systems are a suitable framework for representing qualitative knowledge, either provided by human experts (knowledge-based fuzzy control) or automatically acquired from data (rule induction, learning). In the latter case, fuzzy clustering algorithms are often used to partition data into groups of similar objects. Fuzzy sets and if–then rules are then induced from the obtained partitioning, see Figure 1.2. In this way, a compact, qualitative summary of a large amount of possibly high-dimensional data is generated. To increase the flexibility and representational power, local regression models can be used in the conclusion part of the rules (the so-called Takagi–Sugeno fuzzy system).

Artificial Neural Networks are simple models imitating the function of biological neural systems. While in fuzzy logic systems, information is represented explicitly in the form of if–then rules, in neural networks, it is implicitly 'coded' in the network and its parameters. In contrast to knowledge-based techniques, no explicit knowledge is needed for the application of neural nets. Their main strength is the ability to learn complex functional relations by generalizing from a limited amount of training data. Neural nets can be used, for instance,

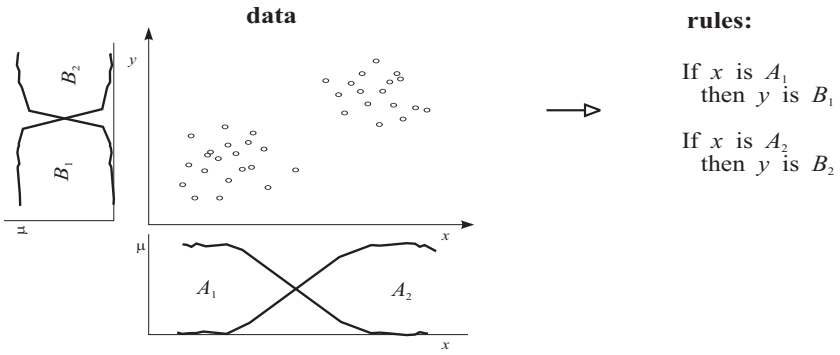


Figure 1.2.: Fuzzy clustering can be used to extract qualitative if–then rules from numerical data.

as (black-box) models for nonlinear, multivariable static and dynamic systems and can be trained by using input–output data observed on the system.

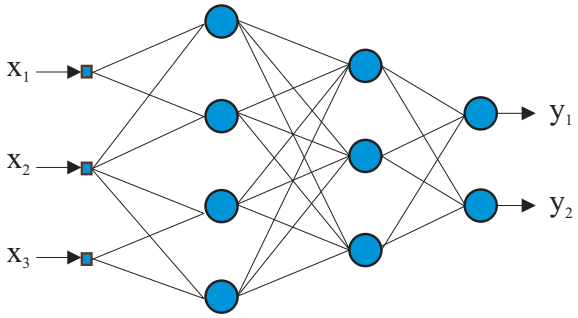


Figure 1.3.: Multi-layer artificial neural network.

Figure 1.3 shows the most common artificial feedforward neural network, which consists of several layers of simple nonlinear processing elements called neurons, inter-connected through adjustable weights. The information relevant to the input–output mapping of the net is stored in these weights. There are many other network architectures, such as multi-layer recurrent nets, Hopfield networks, or self-organizing maps. Neural networks and fuzzy systems are often combined in *neuro-fuzzy* systems, which effectively integrate qualitative rule-based techniques with data-driven learning.

Genetic algorithms are randomized optimization techniques inspired by the principles of natural evolution and survival of the fittest. Candidate solutions to the problem at hand are coded as strings of binary or real numbers. The fitness (quality, performance) of the individual solutions is evaluated by means of a fitness function, which is defined externally by the user or another higher-level algorithm. The fittest individuals in the population of solutions are reproduced, using genetic operators like the crossover and mutation. In this way, a new, generation of fitter individuals is obtained and the whole cycle starts again (Figure 1.4). Genetic algorithms proved to be effective in searching high-dimensional spaces and have found applications in a large number of domains, including the optimization of model or controller structures, the tuning of parameters in nonlinear systems, etc. Genetic algorithms are not addressed in this course.

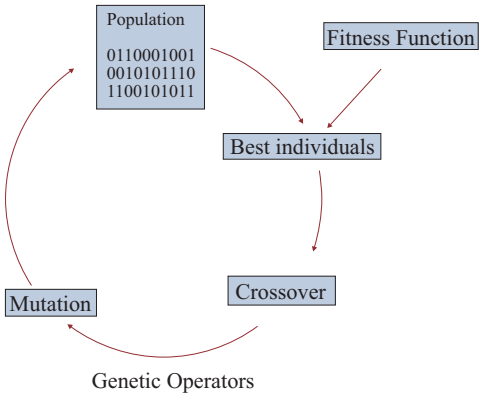


Figure 1.4.: Genetic algorithms are based on a simplified simulation of the natural evolution cycle.

1.4. Organization of the Book

The material is organized in eight chapters. In Chapter 2, the basics of fuzzy set theory are explained. Chapter 3 then presents various types of fuzzy systems and their application in dynamic modeling. Fuzzy set techniques can be useful in data analysis and pattern recognition. To this end, Chapter 4 presents the basic concepts of fuzzy clustering, which can be used as data-driven techniques for the construction of fuzzy models from data. These data-driven construction techniques are addressed in Chapter 5. Controllers can also be designed without using a process model. Chapter 6 is devoted to model-free knowledge-based design of fuzzy controllers. In Chapter 7, artificial neural networks are explained in terms of their architectures and training methods. Neural and fuzzy models can be used to design a controller or can become part of a model-based control scheme, as explain in Chapter 8. Reinforcement learning control is the subject of Chapter 9.

Three appendices have been included to provide background material on ordinary set theory (Appendix A), MATLAB code for some of the presented methods and algorithms (Appendix B) and a list of symbols used throughout the text (Appendix C).

It has been one of the author's aims to present the new material (fuzzy and neural techniques, reinforcement learning) in such a way that no prior knowledge about these subjects is required for an understanding of the text. It is assumed, however, that the reader has some basic knowledge of mathematical analysis (univariate and multivariate functions), linear algebra (system of linear equations, least-square solution) and systems and control theory (dynamic systems, state-feedback, PID control, linearization). Sections marked with an asterisk (*) present additional material, which is not compulsory for the examination.

1.5. WEB and Matlab Support

The material presented in the book is supported by a Web page containing basic information about the course 'Knowledge-Based Control Systems' (SC42050) given at the Delft University of Technology, as well as a number of items for download (MATLAB

tools and demos, handouts of lecture transparencies, a sample exam). The URL is (`http://dcsc.tudelft.nl/~sc42050`). Students following the course are encouraged to install MATLAB via the campus licence on their own computers.

1.6. Further Reading

Sources cited throughout the text provide pointers to the literature relevant to the specific subjects addressed. Some general books on intelligent control and its various ingredients include the following ones (among many others):

- Harris, C. J., Moore, C., and Brown, M. (1993). *Intelligent Control, Aspects of Fuzzy Logic and Neural Nets*. World Scientific, Singapore
- Haykin, S. (1994). *Neural Networks*. Macmillan Maxwell International, New York
- Jang, J.-S. R., Sun, C.-T., and Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing; a Computational Approach to Learning and Machine Intelligence*. Prentice-Hall, Upper Saddle River
- Klir, G. J. and Yuan, B. (1995). *Fuzzy sets and fuzzy logic; theory and applications*. Prentice Hall
- Passino, K. M. and Yurkovich, S. (1998). *Fuzzy Control*. Addison-Wesley, Massachusetts, USA
- Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA
- Zurada, J. M., Marks II, R. J., and Robinson, C. J., editors (1994). *Computational Intelligence: Imitating Life*. IEEE Press, Piscataway, NJ

1.7. Acknowledgements

I wish to express my sincere thanks to my colleague Jelmer van Ast, who wrote a large part of Chapter 9, and to Janos Abonyi, Stanimir Mollov and Olaf Wolkenhauer who read parts of the manuscript and contributed by their comments and suggestions. Several students provided feedback on the material and thus helped to improve it. Thanks are also due to Govert Monsees who developed the sliding-mode controller presented in Example 6.2. Figures 6.1, 6.2 and 6.4 were kindly provided by Karl-Erik Årzén.

2. Fuzzy Sets and Relations

This chapter provides a basic introduction to fuzzy sets, fuzzy relations and operations with fuzzy sets. For a more comprehensive treatment see, for instance, (Klir and Folger, 1988; Zimmermann, 1996; Klir and Yuan, 1995).

Zadeh (1965) introduced fuzzy set theory as a mathematical discipline, although the underlying ideas had already been recognized earlier by philosophers and logicians (Pierce, Russel, Łukasiewicz, among others). A comprehensive overview is given in the introduction of the "Readings in Fuzzy Sets for Intelligent Systems", edited by Dubois et al. (1993). A broader interest in fuzzy sets started in the seventies with their application to control and other technical disciplines.

2.1. Fuzzy Sets

In ordinary (non fuzzy) set theory, elements either fully belong to a set or are fully excluded from it. Recall, that the membership $\mu_A(x)$ of x of a classical set A, as a subset of the universe X, is defined by:[1]

$$\mu_A(x) = \begin{cases} 1, & \text{iff} \quad x \in A, \\ 0, & \text{iff} \quad x \notin A\,. \end{cases} \tag{2.1}$$

This means that an element x is either a member of set A ($\mu_A(x) = 1$) or not ($\mu_A(x) = 0$). This strict classification is useful in the mathematics and other sciences that rely on precise definitions. Ordinary set theory complements bi-valent logic in which a statement is either true or false. While in mathematical logic the emphasis is on preserving formal validity and truth under any and every interpretation, in many real-life situations and engineering problems, the aim is to preserve information in the given context. In these situations, it may not be quite clear whether an element belongs to a set or not.

For example, if set A represents PCs which are too expensive for a student's budget, then it is obvious that this set has no clear boundaries. Of course, it could be said that a PC priced at \$2500 is too expensive, but what about PCs priced at \$2495 or \$2502? Are those PCs too expensive or not? Clearly, a boundary could be determined above which a PC is too expensive for the average student, say \$2500, and a boundary below which a PC is certainly not too expensive, say \$1000. Between those boundaries, however, there remains a vague interval in which it is not quite clear whether a PC is too expensive or not. In this interval, a grade could be used to classify the price as partly too expensive. This is where fuzzy sets come in: sets of which the membership has grades in the unit interval [0,1].

A fuzzy set is a set with graded membership in the real interval: $\mu_A(x) \in [0, 1]$. That is, elements can belong to a fuzzy set to a certain degree. As such, fuzzy sets can be used for

[1] A brief summary of basic concepts related to ordinary sets is given in Appendix A.

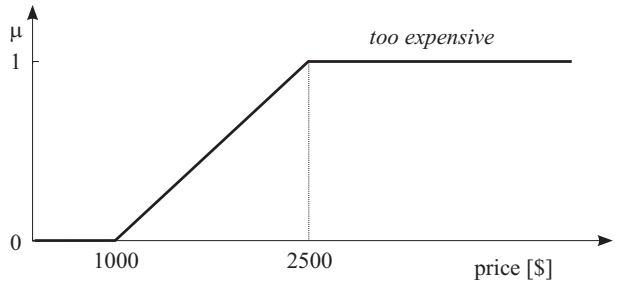


Figure 2.1.: Fuzzy set A representing PCs too expensive for a student's budget.

mathematical representations of vague concepts, such as *low temperature*, *fairly tall person*, *expensive car*, etc.

Definition 2.1 (Fuzzy Set) *A fuzzy set A on universe (domain) X is a set defined by the membership function $\mu_A(x)$ which is a mapping from the universe X into the unit interval:*

$$\mu_A(x) \colon X \to [0,1]\,. \tag{2.2}$$

$\mathcal{F}(X)$ denotes the set of all fuzzy sets on X.

If the value of the membership function, called the membership degree (or grade), equals one, x belongs completely to the fuzzy set. If it equals zero, x does not belong to the set. If the membership degree is between 0 and 1, x is a partial member of the fuzzy set:

$$\mu_A(x) \begin{cases} = 1 & x \text{ is a full member of } A \\ \in (0,1) & x \text{ is a partial member of } A \\ = 0 & x \text{ is not member of } A \end{cases} \tag{2.3}$$

In the literature on fuzzy set theory, ordinary (nonfuzzy) sets are usually referred to as *crisp (or hard) sets*. Various symbols are used to denote membership functions and degrees, such as $\mu_A(x)$, $A(x)$ or just a.

Example 2.1 (Fuzzy Set) Figure 2.1 depicts a possible membership function of a fuzzy set representing PCs *too expensive* for a student's budget.

According to this membership function, if the price is below \$1000 the PC is certainly not too expensive, and if the price is above \$2500 the PC is fully classified as too expensive. In between, an increasing membership of the fuzzy set *too expensive* can be seen. It is not necessary that the membership linearly increases with the price, nor that there is a non-smooth transition from \$1000 to \$2500. Note that in engineering applications the choice of the membership function for a fuzzy set is rather arbitrary.

□

2.2. Properties of Fuzzy Sets

To establish the mathematical framework for computing with fuzzy sets, a number of properties of fuzzy sets need to be defined. This section gives an overview of only the ones

that are strictly needed for the rest of the book. They include the definitions of the height, support, core, α-cut and cardinality of a fuzzy set. In addition, the properties of normality and convexity are introduced. For a more complete treatment see (Klir and Yuan, 1995).

2.2.1. Normal and Subnormal Fuzzy Sets

We learned that the membership of elements in fuzzy sets is a matter of degree. The *height* of a fuzzy set is the largest membership degree among all elements of the universe. Fuzzy sets whose height equals one for at least one element x in the domain X are called *normal* fuzzy sets. The height of *subnormal* fuzzy sets is thus smaller than one for all elements in the domain. Formally we state this by the following definitions.

Definition 2.2 (Height) *The height of a fuzzy set A is the supremum of the membership grades of elements in A:*

$$\operatorname{hgt}(A) = \sup_{x \in X} \mu_A(x)\,. \tag{2.4}$$

For a discrete domain X, the supremum (the least upper bound) becomes the maximum and hence the height is the largest degree of membership for all $x \in X$.

Definition 2.3 (Normal Fuzzy Set) *A fuzzy set A is* normal *if $\exists x \in X$ such that $\mu_A(x) = 1$. Fuzzy sets that are not normal are called* subnormal. *The operator* $\operatorname{norm}(A)$ *denotes normalization of a fuzzy set, i.e., $A' = \operatorname{norm}(A) \Leftrightarrow \mu_{A'}(x) = \mu_A(x)/\operatorname{hgt}(A),\ \forall x$.*

2.2.2. Support, Core and α-cut

Support, core and α-cut are *crisp* sets obtained from a fuzzy set by selecting its elements whose membership degrees satisfy certain conditions.

Definition 2.4 (Support) *The* support *of a fuzzy set A is the crisp subset of X whose elements all have nonzero membership grades:*

$$\operatorname{supp}(A) = \{x \mid \mu_A(x) > 0\}\,. \tag{2.5}$$

Definition 2.5 (Core) *The core of a fuzzy set A is a crisp subset of X consisting of all elements with membership grades equal to one:*

$$\operatorname{core}(A) = \{x \mid \mu_A(x) = 1\}\,. \tag{2.6}$$

In the literature, the core is sometimes also denoted as the kernel, $\ker(A)$. The core of a subnormal fuzzy set is empty.

Definition 2.6 (α-Cut) *The α-cut A_α of a fuzzy set A is the crisp subset of the universe of discourse X whose elements all have membership grades greater than or equal to α:*

$$A_\alpha = \{x \mid \mu_A(x) \geq \alpha\}, \quad \alpha \in [0, 1]\,. \tag{2.7}$$

The α-cut operator is also denoted by $\alpha\text{-cut}(A)$ or $\alpha\text{-cut}(A, \alpha)$. An α-cut A_α is strict if $\mu_A(x) \neq \alpha$ for each $x \in A_\alpha$. The value α is called the α-level.

Figure 2.2 depicts the core, support and α-cut of a fuzzy set.

The core and support of a fuzzy set can also be defined by means of α-cuts:

$$\operatorname{core}(A) = 1\text{-cut}(A) \tag{2.8}$$

$$\operatorname{supp}(A) = 0\text{-cut}(A) \tag{2.9}$$

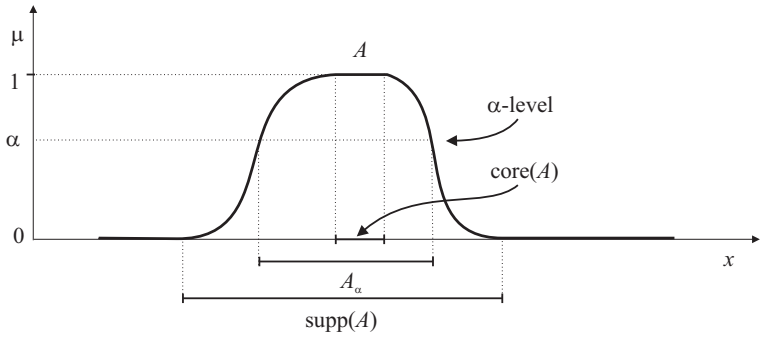


Figure 2.2.: Core, support and α-cut of a fuzzy set.

2.2.3. Convexity and Cardinality

Membership function may be unimodal (with one global maximum) or multimodal (with several maxima). Unimodal fuzzy sets are called convex fuzzy sets. Convexity can also be defined in terms of α-cuts:

Definition 2.7 (Convex Fuzzy Set) *A fuzzy set defined in $\mathbb{R}^n$ is convex if each of its α-cuts is a convex set.*

Figure 2.3 gives an example of a convex and non-convex fuzzy set.

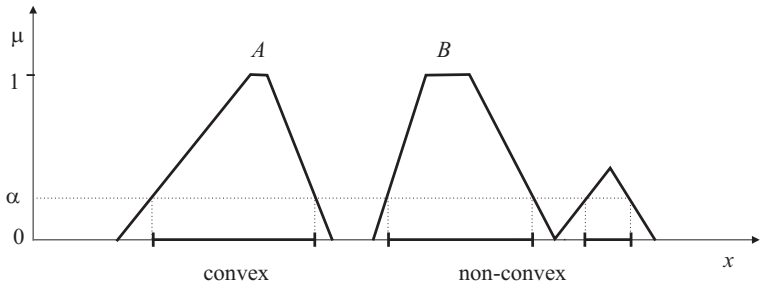


Figure 2.3.: The core of a non-convex fuzzy set is a non-convex (crisp) set.

Example 2.2 (Non-convex Fuzzy Set) Figure 2.4 gives an example of a non-convex fuzzy set representing "high-risk age" for a car insurance policy. Drivers who are too young or too old present higher risk than middle-aged drivers.

□

Definition 2.8 (Cardinality) *Let $A = \{\mu_A(x_i) \mid i = 1, 2, \ldots, n\}$ be a finite discrete fuzzy set. The cardinality of this fuzzy set is defined as the sum of the membership degrees:*

$$|A| = \sum_{i=1}^{n} \mu_A(x_i)\,. \tag{2.10}$$

Cardinality is also denoted by $\mathrm{card}(A)$.

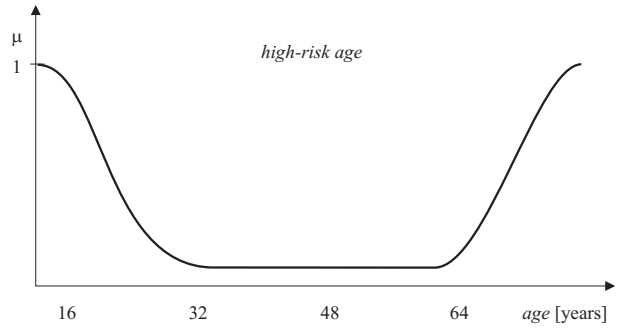


Figure 2.4.: A fuzzy set defining "high-risk age" for a car insurance policy is an example of a non-convex fuzzy set.

2.3. Representations of Fuzzy Sets

There are several ways to define (or represent in a computer) a fuzzy set: through an analytic description of its membership function $\mu_A(x) = f(x)$, as a list of the domain elements end their membership degrees or by means of α-cuts. These possibilities are discussed below.

2.3.1. Similarity-based Representation

Fuzzy sets are often defined by means of the (dis)similarity of the considered object x to a given prototype v of the fuzzy set

$$\mu(x) = \frac{1}{1 + d(x, v)}\,. \tag{2.11}$$

Here, $d(x, v)$ denotes a dissimilarity measure which in metric spaces is typically a distance measure (such as the Euclidean distance). The prototype is a full member (typical element) of the set. Elements whose distance from the prototype goes to zero have membership grades close to one. As the distance grows, the membership decreases. As an example, consider the membership function:

$$\mu_A(x) = \frac{1}{1 + x^2}, \quad x \in \mathbb{R},$$

representing "approximately zero" real numbers.

2.3.2. Parametric Functional Representation

Various forms of parametric membership functions are often used:

- *Trapezoidal* membership function:

$$\mu(x; a, b, c, d) = \max\left(0, \min\left(\frac{x - a}{b - a}, 1, \frac{d - x}{d - c}\right)\right), \tag{2.12}$$

  where a, b, c and d are the coordinates of the trapezoid's apexes. When $b = c$, a *triangular* membership function is obtained.

- *Piece-wise exponential* membership function:

$$\mu(x; c_l, c_r, w_l, w_r) = \begin{cases} \exp(-(\frac{x-c_l}{2w_l})^2), & \text{if } x < c_l, \\ \exp(-(\frac{x-c_r}{2w_r})^2), & \text{if } x > c_r, \\ 1, & \text{otherwise}, \end{cases} \tag{2.13}$$

  where c_l and c_r are the left and right shoulder, respectively, and w_l, w_r are the left and right width, respectively. For $c_l = c_r$ and $w_l = w_r$ the Gaussian membership function is obtained.

Figure 2.5 shows examples of triangular, trapezoidal and bell-shaped (exponential) membership functions. A special fuzzy set is the *singleton set* (fuzzy set representation of a number) defined by:

$$\mu_A(x) = \begin{cases} 1, & \text{if } x = x_0, \\ 0, & \text{otherwise}. \end{cases} \tag{2.14}$$

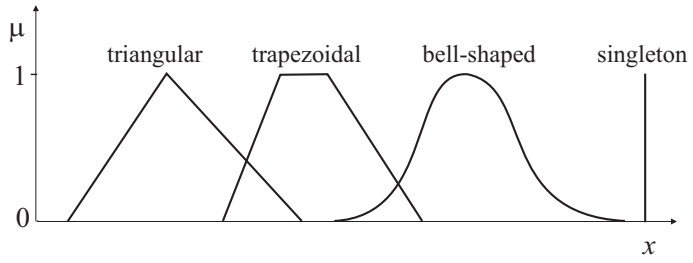


Figure 2.5.: Different shapes of membership functions.

Another special set is the *universal set*, whose membership function equals one for all domain elements:

$$\mu_A(x) = 1, \quad \forall x\,. \tag{2.15}$$

Finally, the term *fuzzy number* is sometimes used to denote a normal, convex fuzzy set which is defined on the real line.

2.3.3. Point-wise Representation

In a discrete set $X = \{x_i \mid i = 1, 2, \ldots, n\}$, a fuzzy set A may be defined by a list of ordered pairs: membership degree/set element:

$$A = \{\mu_A(x_1)/x_1,\ \mu_A(x_2)/x_2,\ \ldots, \mu_A(x_n)/x_n\} = \{\mu_A(x)/x \mid x \in X\}, \tag{2.16}$$

Normally, only elements $x \in X$ with non-zero membership degrees are listed. The following alternatives to the above notation can be encountered:

$$A = \mu_A(x_1)/x_1 + \cdots + \mu_A(x_n)/x_n = \sum_{i=1}^{n} \mu_A(x_i)/x_i \tag{2.17}$$

for finite domains, and

$$A = \int_X \mu_A(x)/x \tag{2.18}$$

for continuous domains. Note that rather than summation and integration, in this context, the $\sum$, $+$ and $\int$ symbols represent a collection (union) of elements.

A pair of vectors (arrays in computer programs) can be used to store discrete membership functions:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n], \quad \boldsymbol{\mu} = [\mu_A(x_1), \mu_A(x_2), \ldots, \mu_A(x_n)]\,. \tag{2.19}$$

Intermediate points can be obtained by interpolation. This representation is often used in commercial software packages. For an equidistant discretization of the domain it is sufficient to store only the membership degrees μ.

2.3.4. Level Set Representation

A fuzzy set can be represented as a list of α levels ($\alpha \in [0,1]$) and their corresponding α-cuts:

$$A = \{\alpha_1/A_{\alpha_1},\ \alpha_2/A_{\alpha_2},\ \ldots, \alpha_n/A_{\alpha_n}\} = \{\alpha/A_{\alpha_n} \mid \alpha \in (0,1)\}, \tag{2.20}$$

The range of α must obviously be discretized. This representation can be advantageous as operations on fuzzy subsets of the same universe can be defined as classical set operations on their level sets. Fuzzy arithmetic can thus be implemented by means of interval arithmetic, etc. In multidimensional domains, however, the use of the level-set representation can be computationally involved.

Example 2.3 (Fuzzy Arithmetic) Using the level-set representation, results of arithmetic operations with fuzzy numbers can be obtained as a collection standard arithmetic operations on their α-cuts. As an example consider addition of two fuzzy numbers A and B defined on the real line:

$$A + B = \{\alpha/(A_{\alpha_n} + B_{\alpha_n}) \mid \alpha \in (0,1)\}, \tag{2.21}$$

where $A_{\alpha_n} + B_{\alpha_n}$ is the addition of two intervals.

□

2.4. Operations on Fuzzy Sets

Definitions of set-theoretic operations such as the complement, union and intersection can be extended from ordinary set theory to fuzzy sets. As membership degrees are no longer restricted to $\{0,1\}$ but can have any value in the interval $[0,1]$, these operations cannot be uniquely defined. It is clear, however, that the operations for fuzzy sets must give correct results when applied to ordinary sets (an ordinary set can be seen as a special case of a fuzzy set).

This section presents the basic definitions of fuzzy intersection, union and complement, as introduced by Zadeh. General intersection and union operators, called triangular norms (t-norms) and triangular conorms (t-conorms), respectively, are given as well. In addition, operations of projection and cylindrical extension, related to multi-dimensional fuzzy sets, are given.

2.4.1. Complement, Union and Intersection

Definition 2.9 (Complement of a Fuzzy Set) *Let A be a fuzzy set in X. The complement of A is a fuzzy set, denoted $\bar{A}$, such that for each $x \in X$:*

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)\,. \tag{2.22}$$

Figure 2.6 shows an example of a fuzzy complement in terms of membership functions. Besides this operator according to Zadeh, other complements can be used. An example is

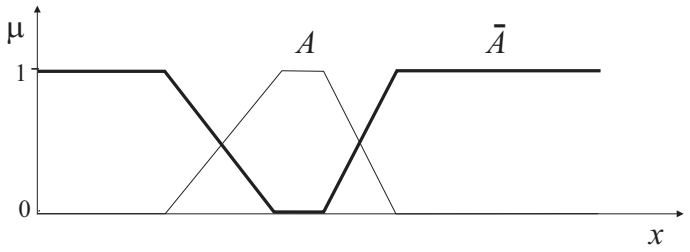


Figure 2.6.: Fuzzy set and its complement $\bar{A}$ in terms of their membership functions.

the λ-complement according to Sugeno (1977):

$$\mu_{\bar{A}}(x) = \frac{1 - \mu_A(x)}{1 + \lambda\mu_A(x)} \tag{2.23}$$

where $\lambda > 0$ is a parameter.

Definition 2.10 (Intersection of Fuzzy Sets) *Let A and B be two fuzzy sets in X. The intersection of A and B is a fuzzy set C, denoted $C = A \cap B$, such that for each $x \in X$:*

$$\mu_C(x) = \min[\mu_A(x), \mu_B(x)]\,. \tag{2.24}$$

The minimum operator is also denoted by '$\wedge$', i.e., $\mu_C(x) = \mu_A(x) \wedge \mu_B(x)$. Figure 2.7 shows an example of a fuzzy intersection in terms of membership functions.

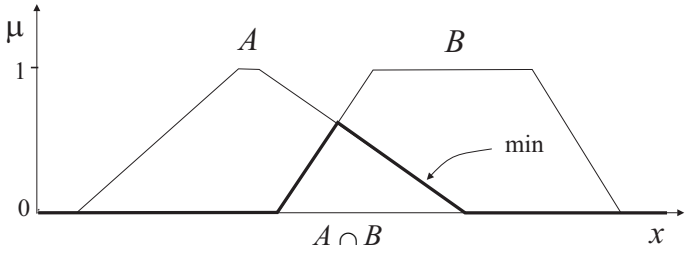


Figure 2.7.: Fuzzy intersection $A \cap B$ in terms of membership functions.

Definition 2.11 (Union of Fuzzy Sets) *Let A and B be two fuzzy sets in X. The union of A and B is a fuzzy set C, denoted $C = A \cup B$, such that for each $x \in X$:*

$$\mu_C(x) = \max[\mu_A(x), \mu_B(x)]\,. \tag{2.25}$$

The maximum operator is also denoted by '$\vee$', i.e., $\mu_C(x) = \mu_A(x) \vee \mu_B(x)$. Figure 2.8 shows an example of a fuzzy union in terms of membership functions.

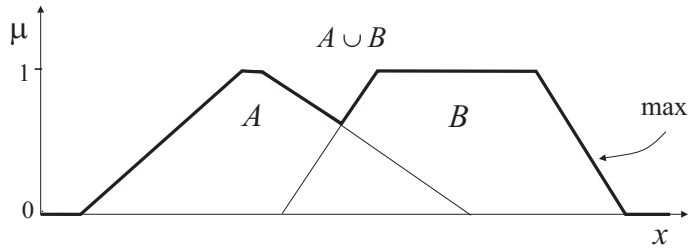


Figure 2.8.: Fuzzy union $A \cup B$ in terms of membership functions.

2.4.2. T-norms and T-conorms

Fuzzy intersection of two fuzzy sets can be specified in a more general way by a binary operation on the unit interval, i.e., a function of the form:

$$T\colon [0,1] \times [0,1] \to [0,1] \tag{2.26}$$

In order for a function T to qualify as a fuzzy intersection, it must have appropriate properties. Functions known as t-norms (triangular norms) posses the properties required for the intersection. Similarly, functions called t-conorms can be used for the fuzzy union.

Definition 2.12 (t-Norm/Fuzzy Intersection) *A t-norm T is a binary operation on the unit interval that satisfies at least the following axioms for all $a, b, c \in [0,1]$ (Klir and Yuan, 1995):*

$$\begin{array}{ll} T(a,1) = a & \textit{(boundary condition)}, \\ b \leq c \quad \textit{implies} \quad T(a,b) \leq T(a,c) & \textit{(monotonicity)}, \\ T(a,b) = T(b,a) & \textit{(commutativity)}, \\ T(a,T(b,c)) = T(T(a,b),c) & \textit{(associativity)}\,. \end{array} \tag{2.27}$$

Some frequently used t-norms are:

standard (Zadeh) intersection:	$T(a,b) = \min(a,b)$
algebraic product (probabilistic intersection):	$T(a,b) = ab$
Łukasiewicz (bold) intersection:	$T(a,b) = \max(0, a+b-1)$

The minimum is the largest t-norm (intersection operator). For our example shown in Figure 2.7 this means that the membership functions of fuzzy intersections $A \cap B$ obtained with other t-norms are all below the bold membership function (or partly coincide with it).

Definition 2.13 (t-Conorm/Fuzzy Union) *A t-conorm S is a binary operation on the unit interval that satisfies at least the following axioms for all $a, b, c \in [0,1]$ (Klir and Yuan, 1995):*

$$\begin{array}{ll} S(a,0) = a & \textit{(boundary condition)}, \\ b \leq c \quad \textit{implies} \quad S(a,b) \leq S(a,c) & \textit{(monotonicity)}, \\ S(a,b) = S(b,a) & \textit{(commutativity)}, \\ S(a,S(b,c)) = S(S(a,b),c) & \textit{(associativity)}\,. \end{array} \tag{2.28}$$

Some frequently used t-conorms are:

standard (Zadeh) union:	$S(a,b) = \max(a,b)$,
algebraic sum (probabilistic union):	$S(a,b) = a + b - ab$,
Łukasiewicz (bold) union:	$S(a,b) = \min(1, a+b)$.

The maximum is the smallest t-conorm (union operator). For our example shown in Figure 2.8 this means that the membership functions of fuzzy unions $A \cup B$ obtained with other t-conorms are all above the bold membership function (or partly coincide with it).

2.4.3. Projection and Cylindrical Extension

Projection reduces a fuzzy set defined in a multi-dimensional domain (such as $\mathbb{R}^2$ to a fuzzy set defined in a lower-dimensional domain (such as $\mathbb{R}$). *Cylindrical extension* is the opposite operation, i.e., the extension of a fuzzy set defined in low-dimensional domain into a higher-dimensional domain. Formally, these operations are defined as follows:

Definition 2.14 (Projection of a Fuzzy Set) *Let $U \subseteq U_1 \times U_2$ be a subset of a Cartesian product space, where U_1 and U_2 can themselves be Cartesian products of lower-dimensional domains. The projection of fuzzy set A defined in U onto U_1 is the mapping* $\text{proj}_{U_1} : \mathcal{F}(U) \to \mathcal{F}(U_1)$ *defined by*

$$\text{proj}_{U_1}(A) = \left\{ \sup_{U_2} \mu_A(u)/u_1 \middle| u_1 \in U_1 \right\} . \tag{2.29}$$

The projection mechanism eliminates the dimensions of the product space by taking the supremum of the membership function for the dimension(s) to be eliminated.

Example 2.4 (Projection) Assume a fuzzy set A defined in $U \subset X \times Y \times Z$ with $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$ and $Z = \{z_1, z_2\}$, as follows:

$$\begin{aligned} A \;=\; & \{\mu_1/(x_1,y_1,z_1),\ \mu_2/(x_1,y_2,z_1),\ \mu_3/(x_2,y_1,z_1), \\ & \mu_4/(x_2,y_2,z_1),\ \mu_5/(x_2,y_2,z_2)\} \end{aligned} \tag{2.30}$$

Let us compute the projections of A onto X, Y and $X \times Y$:

$$\text{proj}_X(A) = \{\max(\mu_1,\mu_2)/x_1,\ \max(\mu_3,\mu_4,\mu_5)/x_2\}, \tag{2.31}$$

$$\text{proj}_Y(A) = \{\max(\mu_1,\mu_3)/y_1,\ \max(\mu_2,\mu_4,\mu_5)/y_2\}, \tag{2.32}$$

$$\begin{aligned} \text{proj}_{X\times Y}(A) = \; & \{\mu_1/(x_1,y_1),\ \mu_2/(x_1,y_2), \\ & \mu_3/(x_2,y_1),\ \max(\mu_4,\mu_5)/(x_2,y_2)\} . \end{aligned} \tag{2.33}$$

□

Projections from $\mathbb{R}^2$ to $\mathbb{R}$ can easily be visualized, see Figure 2.9.

Definition 2.15 (Cylindrical Extension) *Let $U \subseteq U_1 \times U_2$ be a subset of a Cartesian product space, where U_1 and U_2 can themselves be Cartesian products of lower-dimensional domains. The cylindrical extension of fuzzy set A defined in U_1 onto U is the mapping* $\text{ext}_U : \mathcal{F}(U_1) \to \mathcal{F}(U)$ *defined by*

$$\text{ext}_U(A) = \left\{ \mu_A(u_1)/u \middle| u \in U \right\} . \tag{2.34}$$

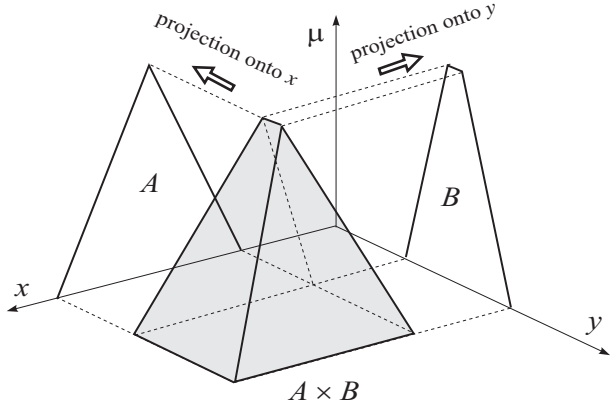


Figure 2.9.: Example of projection from $\mathbb{R}^2$ to $\mathbb{R}$.

Cylindrical extension thus simply replicates the membership degrees from the existing dimensions into the new dimensions. Figure 2.10 depicts the cylindrical extension from $\mathbb{R}$ to $\mathbb{R}^2$.

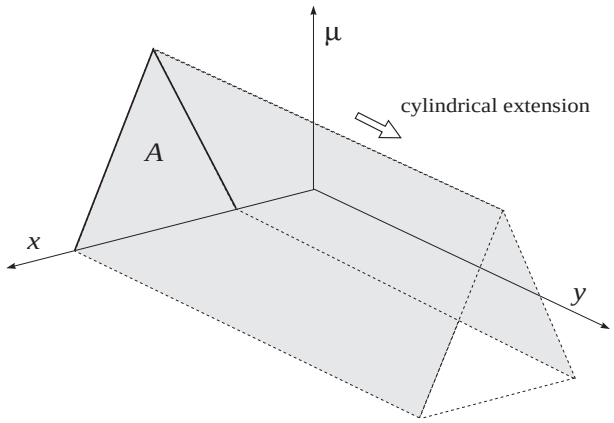


Figure 2.10.: Example of cylindrical extension from $\mathbb{R}$ to $\mathbb{R}^2$.

It is easy to see that projection leads to a loss of information, thus for A defined in $X^n \subset X^m$ $(n < m)$ it holds that:

$$A = \text{proj}_{X^n}(\text{ext}_{X^m}(A)), \tag{2.35}$$

but

$$A \neq \text{ext}_{X^m}(\text{proj}_{X^n}(A))\,. \tag{2.36}$$

Verify this for the fuzzy sets given in Example 2.4 as an exercise.

2.4.4. Operations on Cartesian Product Domains

Set-theoretic operations such as the union or intersection applied to fuzzy sets defined in different domains result in a multi-dimensional fuzzy set in the Cartesian product of those domains. The operation is in fact performed by first extending the original fuzzy sets into

the Cartesian product domain and then computing the operation on those multi-dimensional sets.

Example 2.5 (Cartesian-Product Intersection) Consider two fuzzy sets A_1 and A_2 defined in domains X_1 and X_2, respectively. The intersection $A_1 \cap A_2$, also denoted by $A_1 \times A_2$ is given by:

$$A_1 \times A_2 = \text{ext}_{X_2}(A_1) \cap \text{ext}_{X_1}(A_2)\,. \tag{2.37}$$

This cylindrical extension is usually considered implicitly and it is not stated in the notation:

$$\mu_{A_1 \times A_2}(x_1, x_2) = \mu_{A_1}(x_1) \wedge \mu_{A_2}(x_2)\,. \tag{2.38}$$

Figure 2.11 gives a graphical illustration of this operation.

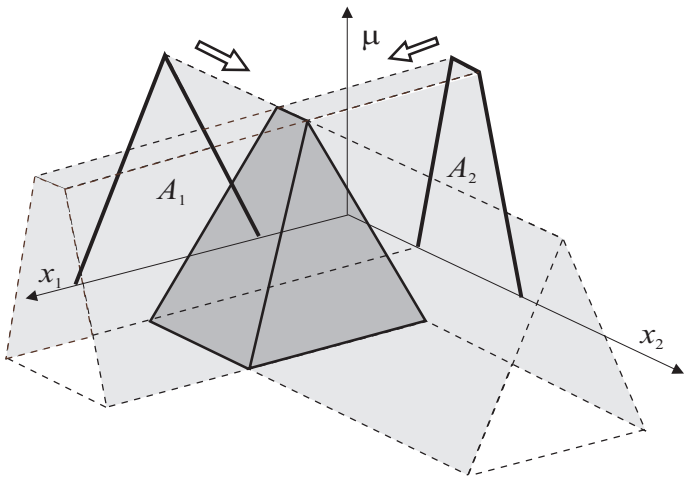


Figure 2.11.: Cartesian-product intersection.

□

2.4.5. Linguistic Hedges

Fuzzy sets can be used to represent qualitative linguistic terms (notions) like “short”, “long”, “expensive”, etc. in terms of membership functions define in numerical domains (distance, price, etc.).

By means of *linguistic hedges* (linguistic modifiers) the meaning of these terms can be modified without redefining the membership functions. Examples of hedges are: *very*, *slightly*, *more or less*, *rather*, etc. Hedge “very”, for instance, can be used to change “expensive” to “very expensive”.

Two basic approaches to the implementation of linguistic hedges can be distinguished: *powered* hedges and *shifted* hedges. Powered hedges are implemented by functions operating on the membership degrees of the linguistic terms (Zimmermann, 1996). For instance, the hedge *very* squares the membership degrees of the term which meaning it modifies, i.e.,

$\mu_{\text{very } A}(x) = \mu_A^2(x)$. Shifted hedges (Lakoff, 1973), on the other hand, shift the membership functions along their domains. Combinations of the two approaches have been proposed as well (Novák, 1989; Novák, 1996).

Example 2.6 Consider three fuzzy sets *Small*, *Medium* and *Big* defined by triangular membership functions. Figure 2.12 shows these membership functions (solid line) along with modified membership functions "more or less small", "nor very small" and "rather big" obtained by applying the hedges in Table 2.1. In this table, A stands for the fuzzy sets and

linguistic hedge	operation	linguistic hedge	operation
very A	μ_A^2	more or less A	$\sqrt{\mu_A}$
not very A	$1 - \mu_A^2$	rather A	int (μ_A)

Table 2.1.: Linguistic hedges

"int" denotes the contrast intensification operator given by:

$$\text{int}\,(\mu_A) = \begin{cases} 2\mu_A^2, & \mu_A \leq 0.5 \\ 1 - 2(1-\mu_A)^2 & \text{otherwise.} \end{cases}$$

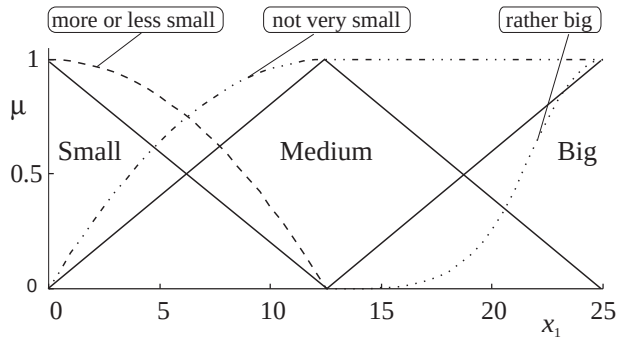


Figure 2.12.: Reference fuzzy sets and their modifications by some linguistic hedges.

□

2.5. Fuzzy Relations

A fuzzy relation is a fuzzy set in the Cartesian product $X_1 \times X_2 \times \cdots \times X_n$. The membership grades represent the degree of association (correlation) among the elements of the different domains X_i.

Definition 2.16 (Fuzzy Relation) *An n-ary fuzzy relation is a mapping*

$$R\colon X_1 \times X_2 \times \cdots \times X_n \to [0,1], \tag{2.39}$$

which assigns membership grades to all n-tuples $(x_1, x_2, \ldots, x_n)$ *from the Cartesian product* $X_1 \times X_2 \times \cdots \times X_n$.

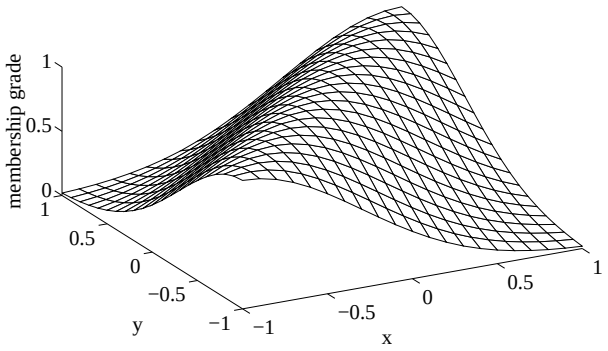


Figure 2.13.: Fuzzy relation $\mu_R(x, y) = e^{-(x-y)^2}$.

For computer implementations, R is conveniently represented as an n-dimensional array: $R = [r_{i_1,i_2,\ldots,i_n}]$.

Example 2.7 (Fuzzy Relation) Consider a fuzzy relation R describing the relationship $x \approx y$ ("x is approximately equal to y") by means of the following membership function $\mu_R(x, y) = e^{-(x-y)^2}$. Figure 2.13 shows a mesh plot of this relation.

□

2.6. Relational Composition

The *composition* is defined as follows (Zadeh, 1973): suppose there exists a fuzzy relation R in $X \times Y$ and A is a fuzzy set in X. Then, fuzzy subset B of Y can be induced by A through the composition of A and R:

$$B = A \circ R\,. \tag{2.40}$$

The composition is defined by:

$$B = \mathrm{proj}_Y\big(R \cap \mathrm{ext}_{X\times Y}(A)\big)\,. \tag{2.41}$$

The composition can be regarded in two phases: *combination* (intersection) and *projection*. Zadeh proposed to use *sup-min* composition. Assume that A is a fuzzy set with membership function $\mu_A(x)$ and R is a fuzzy relation with membership function $\mu_R(x, y)$:

$$\mu_B(y) = \sup_x \min\big(\mu_A(x), \mu_R(x, y)\big), \tag{2.42}$$

where the cylindrical extension of A into $X \times Y$ is implicit and *sup* and *min* represent the projection and combination phase, respectively. In a more general form of the composition, a t-norm T is used for the intersection:

$$\mu_B(y) = \sup_x \mathrm{T}\big(\mu_A(x), \mu_R(x, y)\big)\,. \tag{2.43}$$

Example 2.8 (Relational Composition) Consider a fuzzy relation R which represents the relationship "x is *approximately equal* to y":

$$\mu_R(x,y) = \max(1 - 0.5 \cdot |x-y|, 0)\,. \tag{2.44}$$

Further, consider a fuzzy set A *"approximately 5"*:

$$\mu_A(x) = \max(1 - 0.5 \cdot |x-5|, 0)\,. \tag{2.45}$$

Suppose that R and A are discretized with $x, y = 0, 1, 2, \ldots$, in $[0, 10]$. Then, the composition is:

$$\mu_B(y) = \overbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{2} \\ 1 \\ \frac{1}{2} \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}^{\mu_A(x)} \circ \overbrace{\begin{pmatrix} 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 1 \end{pmatrix}}^{\mu_R(x,y)} =$$

$$= \max_x \overbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}^{\min(\mu_A(x), \mu_R(x,y))} =$$

$$= \overbrace{\begin{pmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \end{pmatrix}}^{\max_x \min(\mu_A(x), \mu_R(x,y))}$$

This resulting fuzzy set, defined in Y can be interpreted as *"approximately 5"*. Note, however, that it is broader (more uncertain) than the set from which it was induced. This is because the uncertainty in the input fuzzy set was combined with the uncertainty in the relation.

□

2.7. Summary and Concluding Remarks

Fuzzy sets are sets without sharp boundaries: membership of a fuzzy set is a real number in the interval $[0, 1]$. Various properties of fuzzy sets and operations on fuzzy sets have been introduced. Relations are multi-dimensional fuzzy sets where the membership grades represent the degree of association (correlation) among the elements of the different domains. The composition of relations, using projection and cylindrical extension is an important concept for fuzzy logic and approximate reasoning, which are addressed in the following chapter.

2.8. Problems

1. What is the difference between the membership function of an ordinary set and of a fuzzy set?

2. Consider fuzzy set C defined by its membership function $\mu_C(x)\colon \mathbb{R} \to [0, 1]$: $\mu_C(x) = 1/(1 + |x|)$. Compute the α-cut of C for $\alpha = 0.5$.

3. Consider fuzzy sets A and B such that $\text{core}(A) \cap \text{core}(B) = \emptyset$. Is fuzzy set $C = A \cap B$ normal? What condition must hold for the supports of A and B such that $\text{card}(C) > 0$ always holds?

4. Consider fuzzy set A defined in $X \times Y$ with $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$:

$$A = \{0.1/(x_1, y_1),\ 0.2/(x_1, y_2),\ 0.7/(x_2, y_1), 0.9/(x_2, y_2)\}$$

   Compute the projections of A onto X and Y.

5. Compute the cylindrical extension of fuzzy set $A = \{0.3/x_1,\ 0.4/x_2\}$ into the Cartesian product domain $\{x_1, x_2\} \times \{y_1, y_2\}$.

6. For fuzzy sets $A = \{0.1/x_1,\ 0.6/x_2\}$ and $B = \{1/y_1,\ 0.7/y_2\}$ compute the union $A \cup B$ and the intersection $A \cap B$. Use the Zadeh's operators (max, min).

7. Given is a fuzzy relation $R\colon X \times Y \to [0, 1]$:

$$R =$$

	y_1	y_2	y_3
x_1	0.7	0.3	0.1
x_2	0.4	0.8	0.2
x_3	0.1	0.2	0.9

   and a fuzzy set $A = \{0.1/x_1,\ 1/x_2,\ 0.4/x_3\}$. Compute fuzzy set $B = A \circ R$, where '$\circ$' is the max-min composition operator.

8. Prove that the following De Morgan law $\overline{(A \cup B)} = \bar{A} \cap \bar{B}$ is true for fuzzy sets A and B, when using the Zadeh's operators for union, intersection and complement.

3. Fuzzy Systems

A static or dynamic system which makes use of fuzzy sets and of the corresponding mathematical framework is called a *fuzzy system*. Fuzzy sets can be involved in a system[1] in a number of ways, such as:

- *In the description of the system.* A system can be defined, for instance, as a collection of if-then rules with fuzzy predicates, or as a fuzzy relation. An example of a fuzzy rule describing the relationship between a heating power and the temperature trend in a room may be:

  If *the heating power is high* **then** *the temperature will increase fast.*

- *In the specification of the system's parameters.* The system can be defined by an algebraic or differential equation, in which the parameters are fuzzy numbers instead of real numbers. As an example consider an equation: $y = \tilde{3}x_1 + \tilde{5}x_2$, where $\tilde{3}$ and $\tilde{5}$ are fuzzy numbers "about three" and "about five", respectively, defined by membership functions. Fuzzy numbers express the uncertainty in the parameter values.

- *The input, output and state variables of a system may be fuzzy sets.* Fuzzy inputs can be readings from unreliable sensors ("noisy" data), or quantities related to human perception, such as comfort, beauty, etc. Fuzzy systems can process such information, which is not the case with conventional (crisp) systems.

A fuzzy system can simultaneously have several of the above attributes. Fuzzy systems can be regarded as a generalization of interval-valued systems, which are in turn a generalization of crisp systems. This relationship is depicted in Figure 3.1 which gives an example of a crisp function and its interval and fuzzy generalizations. The evaluation of the function for crisp, interval and fuzzy data is schematically depicted.

A function $f\colon X \to Y$ can be regarded as a subset of the Cartesian product $X \times Y$, i.e., as a *relation*. The evaluation of the function for a given input proceeds in three steps (Figure 3.1):

1. Extend the given input into the product space $X \times Y$ (vertical dashed lines).

2. Find the intersection of this extension with the relation (intersection of the vertical dashed lines with the function).

3. Project this intersection onto Y (horizontal dashed lines).

[1] Under "systems" we understand both static functions and dynamic systems. For the sake of simplicity, most examples in this chapter are static systems.

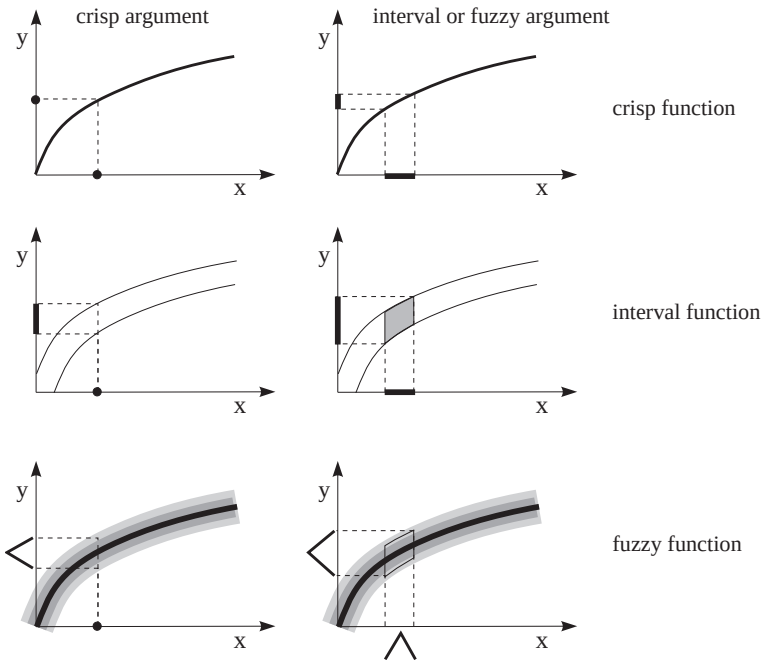


Figure 3.1.: Evaluation of a crisp, interval and fuzzy function for crisp, interval and fuzzy arguments.

This procedure is valid for crisp, interval and fuzzy functions and data. Remember this view, as it will help you to understand the role of fuzzy relations in fuzzy inference.

Most common are fuzzy systems defined by means of if-then rules: *rule-based fuzzy systems*. In the rest of this text we will focus on these systems only. Fuzzy systems can serve different purposes, such as modeling, data analysis, prediction or control. In this text a fuzzy rule-based system is simply called a *fuzzy model*, regardless of its eventual purpose.

3.1. Rule-Based Fuzzy Systems

In rule-based fuzzy systems, the relationships between variables are represented by means of fuzzy if–then rules in the following general form:

If *antecedent proposition* **then** *consequent proposition*.

Fuzzy propositions are statements like "x is big", where "big" is a *linguistic label*, defined by a fuzzy set on the universe of discourse of variable x. Linguistic labels are also referred to as fuzzy constants, fuzzy terms or fuzzy notions. Linguistic modifiers (hedges) can be used to modify the meaning of linguistic labels. For example, the linguistic modifier *very* can be used to change "x is big" to "x is *very* big".

The antecedent proposition is always a fuzzy proposition of the type "$\mathbf{x}$ is A" where $\mathbf{x}$ is a linguistic variable and A is a linguistic constant (term). Depending on the particular structure of the consequent proposition, three main types of models are distinguished:

- *Linguistic fuzzy model* (Zadeh, 1973; Mamdani, 1977), where both the antecedent and consequent are fuzzy propositions. *Singleton* fuzzy model is a special case where the consequents are singleton sets (real constants).

- *Fuzzy relational model* (Pedrycz, 1984; Yi and Chung, 1993), which can be regarded as a generalization of the linguistic model, allowing one particular antecedent proposition to be associated with several different consequent propositions via a fuzzy relation.
- *Takagi–Sugeno (TS) fuzzy model* (Takagi and Sugeno, 1985), where the consequent is a crisp function of the antecedent variables rather than a fuzzy proposition.

These types of fuzzy models are detailed in the subsequent sections.

3.2. Linguistic model

The linguistic fuzzy model (Zadeh, 1973; Mamdani, 1977) has been introduced as a way to capture qualitative knowledge in the form of if–then rules:

$$\mathcal{R}_i\colon\ \textbf{If } \mathbf{x} \text{ is } A_i \textbf{ then } \mathbf{y} \text{ is } B_i, \qquad i = 1, 2, \ldots, K\,. \tag{3.1}$$

Here $\mathbf{x}$ is the input (antecedent) *linguistic variable*, and A_i are the antecedent *linguistic terms* (labels). Similarly, $\mathbf{y}$ is the output (consequent) linguistic variable and B_i are the consequent linguistic terms. The values of $\mathbf{x}$ ($\mathbf{y}$) are generally fuzzy sets, but since a real number is a special case of a fuzzy set (singleton set), these variables can also be real-valued (vectors). The linguistic terms A_i (B_i) are always fuzzy sets

3.2.1. Linguistic Terms and Variables

Linguistic terms can be seen as qualitative values (information granulae) used to describe a particular relationship by linguistic rules. Typically, a set of N linguistic terms $\mathcal{A} = \{A_1, A_2, \ldots, A_N\}$ is defined in the domain of a given variable $\mathbf{x}$. Because this variable assumes linguistic values, it is called a linguistic variable. To distinguish between the linguistic variable and the original numerical variable, the latter one is called the *base variable*.

Definition 3.1 (Linguistic Variable) *A* linguistic variable *L is defined as a quintuple (Klir and Yuan, 1995):*

$$L = (\mathbf{x}, \mathcal{A}, X, g, m), \tag{3.2}$$

where $\mathbf{x}$ *is the* base variable *(at the same time the name of the linguistic variable),* $\mathcal{A} = \{A_1, A_2, \ldots, A_N\}$ *is the set of* linguistic terms*, X is the* domain *(universe of discourse) of* $\mathbf{x}$*, g is a* syntactic rule *for generating linguistic terms and m is a* semantic rule *that assigns to each linguistic term its meaning (a fuzzy set in X).*

Example 3.1 (Linguistic Variable) Figure 3.2 shows an example of a linguistic variable "temperature" with three linguistic terms "low", "medium" and "high". The base variable is the temperature given in appropriate physical units.

□

It is usually required that the linguistic terms satisfy the properties of *coverage* and *semantic soundness* (Pedrycz, 1995).

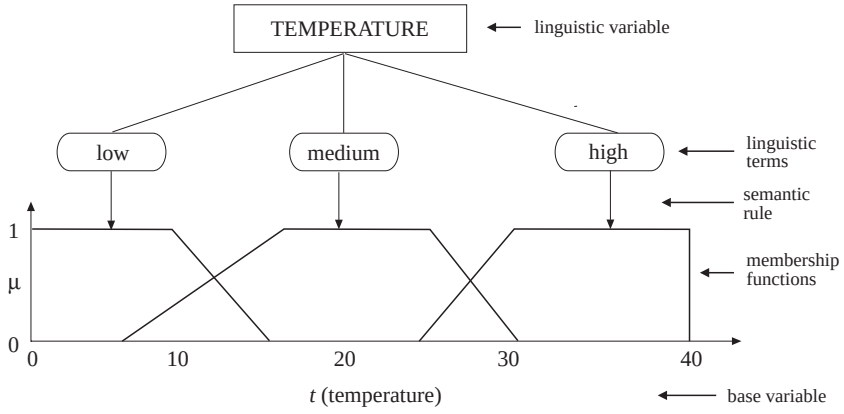


Figure 3.2.: Example of a linguistic variable "temperature" with three linguistic terms.

Coverage

Coverage means that each domain element is assigned to at least one fuzzy set with a nonzero membership degree, i.e.,

$$\forall \mathbf{x} \in X,\ \exists i,\ \mu_{A_i}(\mathbf{x}) > 0\,. \tag{3.3}$$

Alternatively, a stronger condition called ϵ-coverage may be imposed:

$$\forall \mathbf{x} \in X,\ \exists i,\ \mu_{A_i}(\mathbf{x}) > \epsilon, \quad \epsilon \in (0,1)\,. \tag{3.4}$$

For instance, the membership functions in Figure 3.2 satisfy ϵ-coverage for $\epsilon = 0.5$. Clustering algorithms used for the automatic generation of fuzzy models from data, presented in Chapter 4 impose yet a stronger condition:

$$\sum_{i=1}^{N} \mu_{A_i}(\mathbf{x}) = 1, \quad \forall \mathbf{x} \in X, \tag{3.5}$$

meaning that for each $\mathbf{x}$, the sum of membership degrees equals one. Such a set of membership functions is called a (*fuzzy partition*). Chapter 4 gives more details.

Semantic Soundness

Semantic soundness is related to the linguistic meaning of the fuzzy sets. Usually, A_i are convex and normal fuzzy sets, which are sufficiently disjoint, and the number N of subsets per variable is small (say nine at most). The number of linguistic terms and the particular shape and overlap of the membership functions are related to the *granularity* of the information processing within the fuzzy system, and hence also to the level of precision with which a given system can be represented by a fuzzy model. For instance, trapezoidal membership functions, such as those given in Figure 3.2, provide some kind of "information hiding" for data within the cores of the membership functions (e.g., temperatures between 0 and 5 degrees cannot be distinguished, since all are classified as "low" with degree 1). Well-behaved mappings can be accurately represented with a very low granularity.

Membership functions can be defined by the model developer (expert), using prior knowledge, or by experimentation, which is a typical approach in knowledge-based fuzzy control (Driankov et al., 1993). In this case, the membership functions are designed such that they represent the meaning of the linguistic terms in the given context. When input–output data of the system under study are available, methods for constructing or adapting the membership functions from data can be applied, see Chapter 5.

Example 3.2 (Linguistic Model) Consider a simple fuzzy model which qualitatively describes how the heating power of a gas burner depends on the oxygen supply (assuming a constant gas supply). We have a scalar input, the oxygen flow rate (x), and a scalar output, the heating power (y). Define the set of antecedent linguistic terms: $\mathcal{A} = \{Low, OK, High\}$, and the set of consequent linguistic terms: $\mathcal{B} = \{Low, High\}$. The qualitative relationship between the model input and output can be expressed by the following rules:

$\mathcal{R}_1$: **If** O_2 flow rate is *Low* **then** heating power is *Low*.
$\mathcal{R}_2$: **If** O_2 flow rate is *OK* **then** heating power is *High*.
$\mathcal{R}_3$: **If** O_2 flow rate is *High* **then** heating power is *Low*.

The meaning of the linguistic terms is defined by their membership functions, depicted in Figure 3.3. The numerical values along the base variables are selected somewhat arbitrarily. Note that no universal meaning of the linguistic terms can be defined. For this example, it will depend on the type and flow rate of the fuel gas, type of burner, etc. Nevertheless, the qualitative relationship expressed by the rules remains valid.

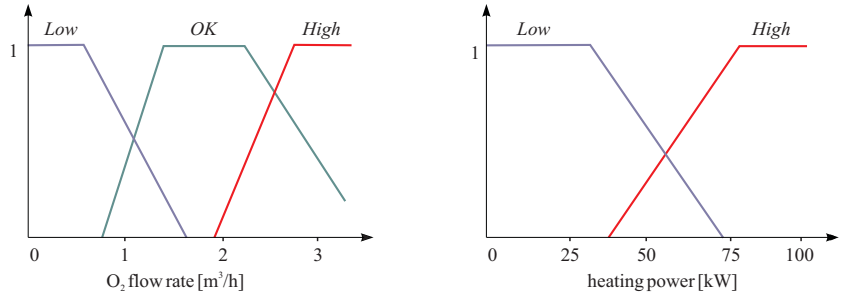


Figure 3.3.: Membership functions.

□

3.2.2. Inference in the Linguistic Model

Inference in fuzzy rule-based systems is the process of deriving an output fuzzy set given the rules and the inputs. The inference mechanism in the linguistic model is based on the *compositional rule of inference* (Zadeh, 1973).

Each rule in (3.1) can be regarded as a fuzzy relation (fuzzy restriction on the simultaneous occurrences of values $\mathbf{x}$ and $\mathbf{y}$): $R\colon (X \times Y) \to [0,1]$ computed by

$$\mu_R(\mathbf{x},\mathbf{y}) = \mathrm{I}(\mu_A(\mathbf{x}), \mu_B(\mathbf{y}))\,. \tag{3.6}$$

For the ease of notation the rule subscript i is dropped. The I operator can be either a fuzzy implication, or a conjunction operator (a t-norm). Note that $I(\cdot,\cdot)$ is computed on the Cartesian product space $X \times Y$, i.e., for all possible pairs of $\mathbf{x}$ and $\mathbf{y}$.

Fuzzy implications are used when the rule (3.1) is regarded as an implication $A_i \to B_i$, i.e., "A_i implies B_i". In classical logic this means that if A holds, B must hold as well for the implication to be true. Nothing can, however, be said about B when A does not hold, and the relationship also cannot be inverted. When using a conjunction, $A \wedge B$, the interpretation of the if-then rules is "it is true that A and B simultaneously hold". This relationship is symmetric (nondirectional) and can be inverted.

Examples of fuzzy implications are the Łukasiewicz implication given by:

$$\mathrm{I}(\mu_A(\mathbf{x}), \mu_B(\mathbf{y})) = \min(1, 1 - \mu_A(\mathbf{x}) + \mu_B(\mathbf{y})), \tag{3.7}$$

or the Kleene–Diene implication:

$$\mathrm{I}(\mu_A(\mathbf{x}), \mu_B(\mathbf{y})) = \max(1 - \mu_A(\mathbf{x}), \mu_B(\mathbf{y}))\,. \tag{3.8}$$

Examples of t-norms are the minimum, often, not quite correctly, called the Mamdani "implication",

$$\mathrm{I}(\mu_A(\mathbf{x}), \mu_B(\mathbf{y})) = \min(\mu_A(\mathbf{x}), \mu_B(\mathbf{y})), \tag{3.9}$$

or the product, also called the Larsen "implication",

$$\mathrm{I}(\mu_A(\mathbf{x}), \mu_B(\mathbf{y})) = \mu_A(\mathbf{x}) \cdot \mu_B(\mathbf{y})\,. \tag{3.10}$$

More details about fuzzy implications and the related operators can be found, for instance, in (Klir and Yuan, 1995; Lee, 1990a,b; Jager, 1995).

The inference mechanism is based on the generalized *modus ponens* rule:

$$\frac{\begin{array}{l}\textbf{If } \mathbf{x} \text{ is } A \textbf{ then } \mathbf{y} \text{ is } B \\ \mathbf{x} \text{ is } A'\end{array}}{\mathbf{y} \text{ is } B'}$$

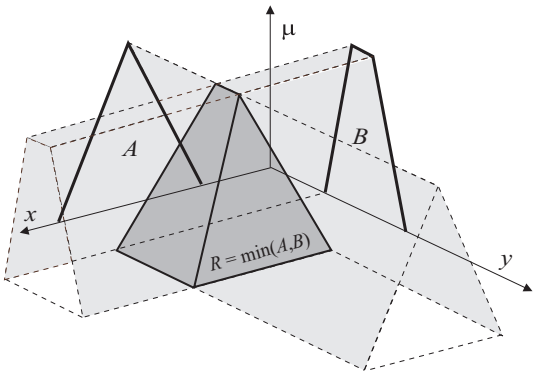
Given the if-then rule and the fact the "$\mathbf{x}$ is A'", the output fuzzy set B' is derived by the relational max-t composition (Klir and Yuan, 1995):

$$B' = A' \circ R\,. \tag{3.11}$$

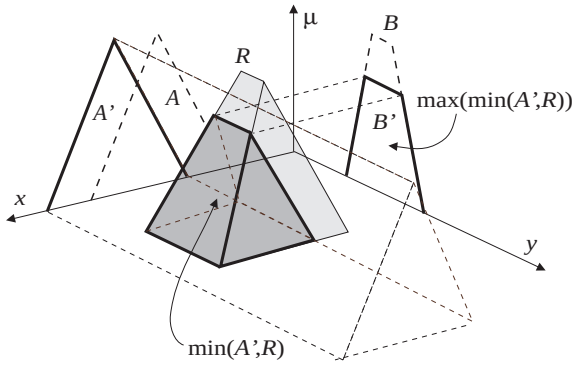
For the minimum t-norm, the max-min composition is obtained:

$$\mu_{B'}(\mathbf{y}) = \max_{X} \min_{X,Y} \left(\mu_{A'}(\mathbf{x}), \mu_R(\mathbf{x}, \mathbf{y})\right). \tag{3.12}$$

Figure 3.4(a) shows an example of fuzzy relation R computed by (3.9). Figure 3.4(b) illustrates the inference of B', given the relation R and the input A', by means of the max-min composition (3.12). One can see that the obtained B' is subnormal, which represents the uncertainty in the input ($A' \neq A$). The relational calculus must be implemented in discrete domains. Let us give an example.



(a) Fuzzy relation (intersection).



(b) Fuzzy inference.

Figure 3.4.: (a) Fuzzy relation representing the rule " If x **is** A **then** y **is** B", (b) the compositional rule of inference.

Example 3.3 (Compositional Rule of Inference) Consider a fuzzy rule

$$\textbf{If x is } A \textbf{ then y is } B$$

with the fuzzy sets:

$$\begin{aligned} A &= \{0/1,\ 0.1/2,\ 0.4/3,\ 0.8/4,\ 1/5\}, \\ B &= \{0/-2,\ 0.6/-1,\ 1/0,\ 0.6/1,\ 0/2\}\,. \end{aligned}$$

Using the minimum t-norm (Mamdani "implication"), the relation R_M representing the fuzzy rule is computed by eq. (3.9):

$$R_M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0.1 & 0.1 & 0 \\ 0 & 0.4 & 0.4 & 0.4 & 0 \\ 0 & 0.6 & 0.8 & 0.6 & 0 \\ 0 & 0.6 & 1 & 0.6 & 0 \end{bmatrix}. \tag{3.13}$$

The rows of this relational matrix correspond to the domain elements of A and the columns to the domain elements of B. Now consider an input fuzzy set to the rule:

$$A' = \{0/1,\ 0.2/2,\ 0.8/3,\ 1/4,\ 0.1/5\}\,. \tag{3.14}$$

The application of the max-min composition (3.12), $B'_M = A' \circ R_M$, yields the following output fuzzy set:

$$B'_M = \{0/-2,\ 0.6/-1,\ 0.8/0,\ 0.6/1,\ 0/2\}\,. \tag{3.15}$$

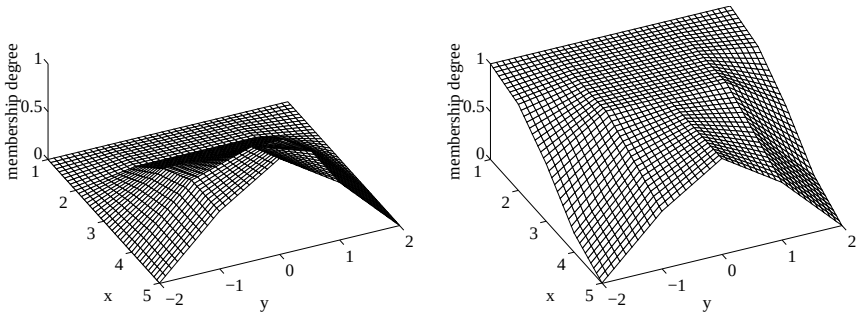
By applying the Łukasiewicz fuzzy implication (3.7), the following relation is obtained:

$$R_L = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0.9 & 1 & 1 & 1 & 0.9 \\ 0.6 & 1 & 1 & 1 & 0.6 \\ 0.2 & 0.8 & 1 & 0.8 & 0.2 \\ 0 & 0.6 & 1 & 0.6 & 0 \end{bmatrix}. \tag{3.16}$$

Using the max-t composition, where the t-norm is the Łukasiewicz (bold) intersection (see Definition 2.12), the inferred fuzzy set $B'_L = A' \circ R_L$ equals:

$$B'_L = \{0.4/-2,\ 0.8/-1,\ 1/0,\ 0.8/1,\ 0.4/2\}\,. \tag{3.17}$$

Note the difference between the relations R_M and R_L, which are also depicted in Figure 3.5. The implication is false (zero entries in the relation) only when A holds and B does not. When A does not hold, the truth value of the implication is 1 regardless of B. The t-norm, however, is false whenever either A or B or both do not hold, and thus represents a bi-directional relation (correlation).



(a) Minimum t-norm. (b) Łukasiewicz implication.

Figure 3.5.: Fuzzy relations obtained by applying a t-norm operator (minimum) and a fuzzy implication (Łukasiewicz).

This difference naturally influences the result of the inference process. Since the input fuzzy set A' is different from the antecedent set A, the derived conclusion B' is in both cases "less certain" than B. The difference is that, with the fuzzy implication, this uncertainty is reflected in the increased membership values for the domain elements that have low or zero membership in B, which means that these output values are possible to a greater degree. However, the t-norm results in decreasing the membership degree of the elements that have high membership in B, which means that these outcomes are less possible. This influences the properties of the two inference mechanisms and the choice of suitable *defuzzification* methods, as discussed later on.

□

The entire rule base (3.1) is represented by aggregating the relations R_i of the individual rules into a single fuzzy relation. If R_i's represent implications, R is obtained by an intersection operator:

$$R = \bigcap_{i=1}^{K} R_i, \quad \text{that is,} \quad \mu_R(\mathbf{x}, \mathbf{y}) = \min_{1 \le i \le K} \mu_{R_i}(\mathbf{x}, \mathbf{y}) \,. \tag{3.18}$$

If I is a t-norm, the aggregated relation R is computed as a union of the individual relations R_i:

$$R = \bigcup_{i=1}^{K} R_i, \quad \text{that is,} \quad \mu_R(\mathbf{x}, \mathbf{y}) = \max_{1 \le i \le K} \mu_{R_i}(\mathbf{x}, \mathbf{y}) \,. \tag{3.19}$$

The output fuzzy set B' is inferred in the same way as in the case of one rule, by using the compositional rule of inference (3.11).

The above representation of a system by the fuzzy relation is called a *fuzzy graph*, and the compositional rule of inference can be regarded as a generalized function evaluation using this graph (see Figure 3.1). The fuzzy relation R, defined on the Cartesian product space of the system's variables $X_1 \times X_2 \times \cdots X_p \times Y$ is a possibility distribution (restriction) of the different input–output tuples $(x_1, x_2, \ldots, x_p, y)$. An α-cut of R can be interpreted as a set of input–output combinations possible to a degree greater or equal to α.

Example 3.4 Let us compute the fuzzy relation for the linguistic model of Example 3.2. First we discretize the input and output domains, for instance: $X = \{0, 1, 2, 3\}$ and $Y = \{0, 25, 50, 75, 100\}$. The (discrete) membership functions are given in Table 3.1 for the antecedent linguistic terms, and in Table 3.2 for the consequent terms.

Table 3.1.: Antecedent membership functions.

	domain element			
linguistic term	0	1	2	3
Low	1.0	0.6	0.0	0.0
OK	0.0	0.4	1.0	0.4
High	0.0	0.0	0.1	1.0

Table 3.2.: Consequent membership functions.

	domain element				
linguistic term	0	25	50	75	100
Low	1.0	1.0	0.6	0.0	0.0
High	0.0	0.0	0.3	0.9	1.0

The fuzzy relations R_i corresponding to the individual rule, can now be computed by using (3.9). For rule $\mathcal{R}_1$, we have $R_1 = \mathit{Low} \times \mathit{Low}$, for rule $\mathcal{R}_2$, we obtain $R_2 = \mathit{OK} \times \mathit{High}$,

and finally for rule $\mathcal{R}_3$, $R_3 = \mathit{High} \times \mathit{Low}$. The fuzzy relation R, which represents the entire rule base, is the union (element-wise maximum) of the relations R_i:

$$\left.\begin{aligned} R_1 &= \begin{bmatrix} 1.0 & 1.0 & 0.6 & 0 & 0 \\ 0.6 & 0.6 & 0.6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ R_2 &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0.4 & 0.4 \\ 0 & 0 & 0.3 & 0.9 & 1.0 \\ 0 & 0 & 0.3 & 0.4 & 0.4 \end{bmatrix} \\ R_3 &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.1 & 0.1 & 0 & 0 \\ 1.0 & 1.0 & 0.6 & 0 & 0 \end{bmatrix} \end{aligned}\right\} \quad R = \begin{bmatrix} 1.0 & 1.0 & 0.6 & 0 & 0 \\ 0.6 & 0.6 & 0.6 & 0.4 & 0.4 \\ 0.1 & 0.1 & 0.3 & 0.9 & 1.0 \\ 1.0 & 1.0 & 0.6 & 0.4 & 0.4 \end{bmatrix}. \tag{3.20}$$

These steps are illustrated in Figure 3.6. For better visualization, the relations are computed with a finer discretization by using the membership functions of Figure 3.3. This example can be run under MATLAB by calling the script `ling`.

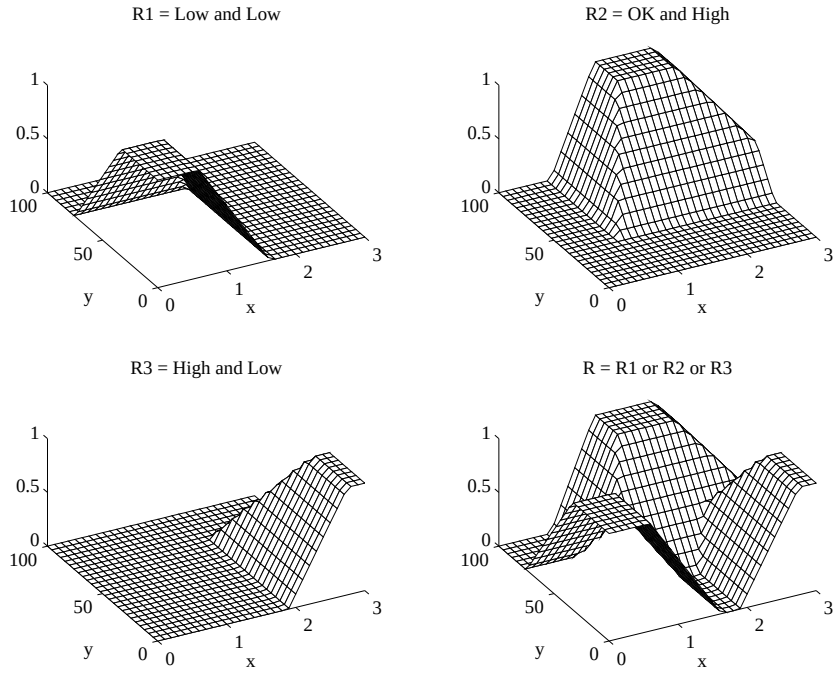


Figure 3.6.: Fuzzy relations R_1, R_2, R_3 corresponding to the individual rules, and the aggregated relation R corresponding to the entire rule base.

Now consider an input fuzzy set to the model, $A' = [1, 0.6, 0.3, 0]$, which can be denoted as *Somewhat Low* flow rate, as it is close to *Low* but does not equal *Low*. The result of max-min composition is the fuzzy set $B' = [1, 1, 0.6, 0.4, 0.4]$, which gives the expected

approximately *Low* heating power. For $A' = [0, 0.2, 1, 0.2]$ (approximately *OK*), we obtain $B' = [0.2, 0.2, 0.3, 0.9, 1]$, i.e., approximately *High* heating power. Verify these results as an exercise. Figure 3.7 shows the fuzzy graph for our example (contours of R, where the shading corresponds to the membership degree).

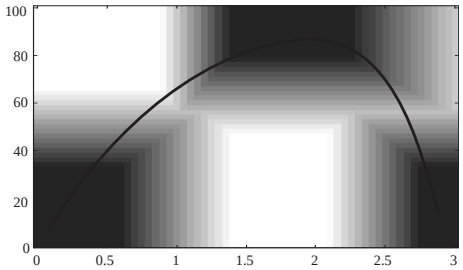


Figure 3.7.: A fuzzy graph for the linguistic model of Example 3.4. Darker shading corresponds to higher membership degree. The solid line is a possible crisp function representing a similar relationship as the fuzzy model.

□

3.2.3. Max-min (Mamdani) Inference

We have seen that a rule base can be represented as a fuzzy relation. The output of a rule-based fuzzy model is then computed by the max-min relational composition. It can be shown that for fuzzy implications with crisp inputs, and for t-norms with both crisp and fuzzy inputs, the reasoning scheme can be simplified, bypassing the relational calculus (Jager, 1995). This is advantageous, as the discretization of domains and storing of the relation R can be avoided. For the t-norm, the simplification results in the well-known scheme, in the literature called the max-min or Mamdani inference, as outlined below.

Suppose an input fuzzy value $\mathbf{x} = A'$, for which the output value B' is given by the relational composition:

$$\mu_{B'}(\mathbf{y}) = \max_X [\mu_{A'}(\mathbf{x}) \wedge \mu_R(\mathbf{x}, \mathbf{y})] \,. \tag{3.21}$$

After substituting for $\mu_R(\mathbf{x}, \mathbf{y})$ from (3.19), the following expression is obtained:

$$\mu_{B'}(\mathbf{y}) = \max_X \left\{ \mu_{A'}(\mathbf{x}) \wedge \max_{1 \le i \le K} [\mu_{A_i}(\mathbf{x}) \wedge \mu_{B_i}(\mathbf{y})] \right\} . \tag{3.22}$$

Since the max and min operation are taken over different domains, their order can be changed as follows:

$$\mu_{B'}(\mathbf{y}) = \max_{1 \le i \le K} \left\{ \max_X [\mu_{A'}(\mathbf{x}) \wedge \mu_{A_i}(\mathbf{x})] \wedge \mu_{B_i}(\mathbf{y}) \right\} . \tag{3.23}$$

Denote $\beta_i = \max_X [\mu_{A'}(\mathbf{x}) \wedge \mu_{A_i}(\mathbf{x})]$ the *degree of fulfillment* of the ith rule's antecedent. The output fuzzy set of the linguistic model is thus:

$$\mu_{B'}(\mathbf{y}) = \max_{1 \le i \le K} [\beta_i \wedge \mu_{B_i}(\mathbf{y})], \quad \mathbf{y} \in Y \,. \tag{3.24}$$

The *max-min* (*Mamdani*) algorithm, is summarized in Algorithm 3.1 and visualized in Figure 3.8.

Algorithm 3.1 *Mamdani (max-min) inference*

1. Compute the degree of fulfillment for each rule by: $\beta_i = \max_X [\mu_{A'}(\mathbf{x}) \wedge \mu_{A_i}(\mathbf{x})]$, $1 \leq i \leq K$. Note that for a singleton set ($\mu_{A'}(\mathbf{x}) = 1$ for $\mathbf{x} = \mathbf{x}_0$ and $\mu_{A'}(\mathbf{x}) = 0$ otherwise) the equation for β_i simplifies to $\beta_i = \mu_{A_i}(\mathbf{x}_0)$.
2. Derive the output fuzzy sets B'_i: $\mu_{B'_i}(\mathbf{y}) = \beta_i \wedge \mu_{B_i}(\mathbf{y}), \quad \mathbf{y} \in Y, \quad 1 \leq i \leq K$.
3. Aggregate the output fuzzy sets B'_i: $\mu_{B'}(\mathbf{y}) = \max_{1 \leq i \leq K} \mu_{B'_i}(\mathbf{y}), \quad \mathbf{y} \in Y$.

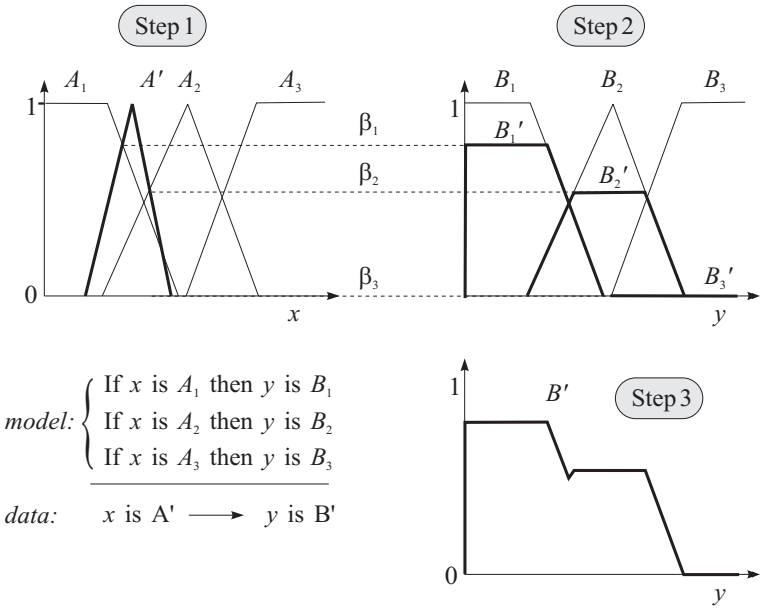


Figure 3.8.: A schematic representation of the Mamdani inference algorithm.

Example 3.5 Let us take the input fuzzy set $A' = [1, 0.6, 0.3, 0]$ from Example 3.4 and compute the corresponding output fuzzy set by the Mamdani inference method. Step 1 yields the following degrees of fulfillment:

$$\begin{aligned}
\beta_1 &= \max_X [\mu_{A'}(x) \wedge \mu_{A_1}(x)] = \max([1, 0.6, 0.3, 0] \wedge [1, 0.6, 0, 0]) = 1.0, \\
\beta_2 &= \max_X [\mu_{A'}(x) \wedge \mu_{A_2}(x)] = \max([1, 0.6, 0.3, 0] \wedge [0, 0.4, 1, 0.4]) = 0.4, \\
\beta_3 &= \max_X [\mu_{A'}(x) \wedge \mu_{A_3}(x)] = \max([1, 0.6, 0.3, 0] \wedge [0, 0, 0.1, 1]) = 0.1\,.
\end{aligned}$$

In step 2, the individual consequent fuzzy sets are computed:

$$\begin{aligned}
B'_1 &= \beta_1 \wedge B_1 = 1.0 \wedge [1, 1, 0.6, 0, 0] = [1, 1, 0.6, 0, 0], \\
B'_2 &= \beta_2 \wedge B_2 = 0.4 \wedge [0, 0, 0.3, 0.9, 1] = [0, 0, 0.3, 0.4, 0.4], \\
B'_3 &= \beta_3 \wedge B_3 = 0.1 \wedge [1, 1, 0.6, 0, 0] = [0.1, 0.1, 0.1, 0, 0]\,.
\end{aligned}$$

Finally, step 3 gives the overall output fuzzy set:

$$B' = \max_{1 \le i \le K} \mu_{B'_i} = [1, 1, 0.6, 0.4, 0.4],$$

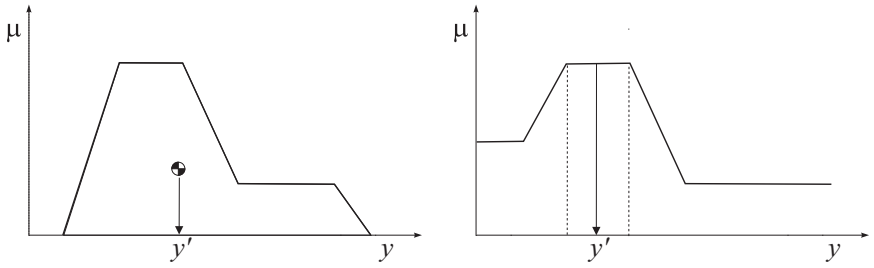
which is identical to the result from Example 3.4. Verify the result for the second input fuzzy set of Example 3.4 as an exercise.

□

From a comparison of the number of operations in examples 3.4 and 3.5, it may seem that the saving with the Mamdani inference method with regard to relational composition is not significant. This is, however, only true for a rough discretization (such as the one used in Example 3.4) and for a small number of inputs (one in this case). Note that the Mamdani inference method does not require any discretization and thus can work with analytically defined membership functions. It also can make use of learning algorithms, as discussed in Chapter 5.

3.2.4. Defuzzification

The result of fuzzy inference is the fuzzy set B'. If a crisp (numerical) output value is required, the output fuzzy set must be *defuzzified*. Defuzzification is a transformation that replaces a fuzzy set by a single numerical value representative of that set. Figure 3.9 shows two most commonly used defuzzification methods: the center of gravity (COG) and the mean of maxima (MOM).



(a) Center of gravity. (b) Mean of maxima.

Figure 3.9.: The center-of-gravity and the mean-of-maxima defuzzification methods.

The COG method calculates numerically the y coordinate of the center of gravity of the fuzzy set B':

$$y' = \mathrm{cog}(B') = \frac{\sum_{j=1}^{F} \mu_{B'}(y_j)\, y_j}{\sum_{j=1}^{F} \mu_{B'}(y_j)} \tag{3.25}$$

where F is the number of elements y_j in Y. Continuous domain Y thus must be discretized to be able to compute the center of gravity.

The MOM method computes the mean value of the interval with the largest membership degree:

$$\mathrm{mom}(B') = \mathrm{cog}\{y \mid \mu_{B'}(y) = \max_{y \in Y} \mu_{B'}(y)\}\,. \tag{3.26}$$

The COG method is used with the Mamdani max-min inference, as it provides interpolation between the consequents, in proportion to the height of the individual consequent sets. This is necessary, as the Mamdani inference method itself does not interpolate, and the use of the MOM method in this case results in a step-wise output. The MOM method is used with the inference based on fuzzy implications, to select the "most possible" output. The inference with implications interpolates, provided that the consequent sets sufficiently overlap (Jager, 1995). The COG method cannot be directly used in this case, because the uncertainty in the output results in an increase of the membership degrees, as shown in Example 3.3. The COG method would give an inappropriate result.

To avoid the numerical integration in the COG method, a modification of this approach called the *fuzzy-mean* defuzzification is often used. The consequent fuzzy sets are first defuzzified, in order to obtain crisp values representative of the fuzzy sets, using for instance the mean-of-maxima method: $b_j = \mathrm{mom}(B_j)$. A crisp output value is then computed by taking a weighted mean of b_j's:

$$y' = \frac{\sum_{j=1}^{M} \omega_j \, b_j}{\sum_{j=1}^{M} \omega_j} \tag{3.27}$$

where M is the number of fuzzy sets B_j and ω_j is the maximum of the degrees of fulfillment β_i over all the rules with the consequent B_j. In terms of the aggregated fuzzy set B', ω_j can be computed by $\omega_j = \mu_{B'}(b_j)$. This method ensures linear interpolation between the b_j's, provided that the antecedent membership functions are piece-wise linear. This is not the case with the COG method, which introduces a nonlinearity, depending on the shape of the consequent functions (Jager et al., 1992). Because the individual defuzzification is done off line, the shape and overlap of the consequent fuzzy sets have no influence, and these sets can be directly replaced by the defuzzified values (singletons), see also Section 3.3. In order to at least partially account for the differences between the consequent fuzzy sets, the weighted fuzzy-mean defuzzification can be applied:

$$y' = \frac{\sum_{j=1}^{M} \gamma_j \, S_j \, b_j}{\sum_{j=1}^{M} \gamma_j \, S_j}, \tag{3.28}$$

where S_j is the area under the membership function of B_j. An advantage of the fuzzy-mean methods (3.27) and (3.28) is that the parameters b_j can be estimated by linear estimation techniques as shown in Chapter 5.

Example 3.6 Consider the output fuzzy set $B' = [0.2, 0.2, 0.3, 0.9, 1]$ from Example 3.4, where the output domain is $Y = [0, 25, 50, 75, 100]$. The defuzzified output obtained by applying formula (3.25) is:

$$y' = \frac{0.2 \cdot 0 + 0.2 \cdot 25 + 0.3 \cdot 50 + 0.9 \cdot 75 + 1 \cdot 100}{0.2 + 0.2 + 0.3 + 0.9 + 1} = 72.12\,.$$

The heating power of the burner, computed by the fuzzy model, is thus 72.12 W.

□

3.2.5. Fuzzy Implication versus Mamdani Inference

A natural question arises: Which inference method is better, or in which situations should one method be preferred to the other? To find an answer, a detailed analysis of the presented methods must be carried out, which is outside the scope of this presentation. One of the distinguishing aspects, however, can be demonstrated by using an example.

Example 3.7 (Advantage of Fuzzy Implications) Consider a rule base of Figure 3.10. Rules R_1 and R_2 represent a simple monotonic (approximately linear) relation between two variables.

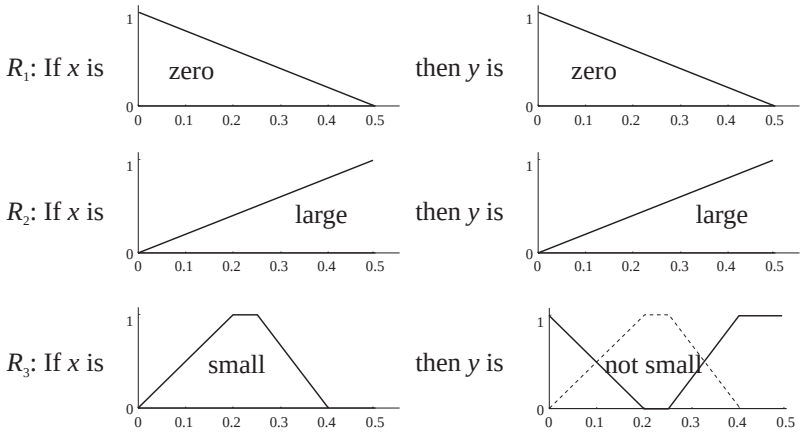
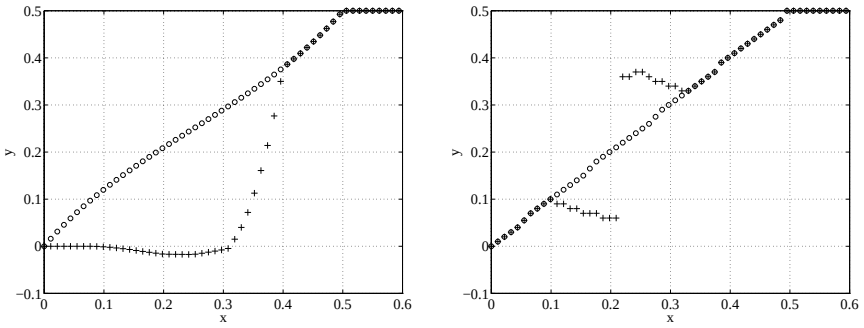


Figure 3.10.: The considered rule base.

This may be, for example, a rule-based implementation of a proportional control law. Rule R_3, "If x is small then y is **not** small", represents a kind of "exception" from the simple relationship defined by interpolation of the previous two rules. In terms of control, such a rule may deal with undesired phenomena, such as static friction. For instance, when controlling an electrical motor with large Coulomb friction, it does not make sense to apply low current if it is not sufficient to overcome the friction, since in that case the motor only consumes energy. These three rules can be seen as a simple example of combining general background knowledge with more specific information in terms of exceptions.

Figure 3.11(a) shows the result for the Mamdani inference method with the COG defuzzification. One can see that the Mamdani method does not work properly. The reason is that the interpolation is provided by the defuzzification method and not by the inference mechanism itself. The presence of the third rule significantly distorts the original, almost linear characteristic, also in the region of x where R_1 has the greatest membership degree. The purpose of avoiding small values of y is not achieved.

Figure 3.11(b) shows the result of logical inference based on the Łukasiewicz implication and MOM defuzzification. One can see that the third rule fulfills its purpose, i.e., forces the fuzzy system to avoid the region of small outputs (around 0.25) for small input values (around 0.25). The exact form of the input–output mapping depends on the choice of the particular inference operators (implication, composition), but the overall behavior remains



(a) Mamdani inference.

(b) Inference with Łukasiewicz implication.

Figure 3.11.: Input-output mapping of the rule base of Figure 3.10 for two different inference methods. Markers 'o' denote the defuzzified output of rules R_1 and R_2 only, markers '+' denote the defuzzified output of the entire rule base.

unchanged.

□

It should be noted, however, that the implication-based reasoning scheme imposes certain requirements on the overlap of the consequent membership functions, which may be hard to fulfill in the case of multi-input rule bases (Jager, 1995). In addition, this method must generally be implemented using fuzzy relations and the compositional rule of inference, which increases the computational demands.

3.2.6. Rules with Several Inputs, Logical Connectives

So far, the linguistic model was presented in a general manner covering both the SISO and MIMO cases. In the MIMO case, all fuzzy sets in the model are defined on vector domains by multivariate membership functions. It is, however, usually, more convenient to write the antecedent and consequent propositions as logical combinations of fuzzy propositions with univariate membership functions. Fuzzy logic operators (connectives), such as the conjunction, disjunction and negation (complement), can be used to combine the propositions.

The connectives *and* and *or* are implemented by t-norms and t-conorms, respectively. There are an infinite number of t-norms and t-conorms, but in practice only a small number of operators are used. Table 3.3 lists the three most common ones.

The choice of t-norms and t-conorms for the logical connectives depends on the meaning and on the context of the propositions. The *max* and *min* operators proposed by Zadeh ignore redundancy, i.e., the combination (conjunction or disjunction) of two identical fuzzy propositions will represent the same proposition:

$$\mu_{A\cap A}(x) = \mu_A(x) \wedge \mu_A(x) = \mu_A(x), \tag{3.29}$$

$$\mu_{A\cup A}(x) = \mu_A(x) \vee \mu_A(x) = \mu_A(x)\,. \tag{3.30}$$

This does not hold for other t-norms and t-conorms. However, when fuzzy propositions are

Table 3.3.: Frequently used operators for the **and** and **or** connectives.

and	or	name
$\min(a, b)$	$\max(a, b)$	Zadeh
$\max(a + b - 1, 0)$	$\min(a + b, 1)$	Łukasiewicz
ab	$a + b - ab$	probability

not equal, but they are correlated or interactive, the use of other operators than *min* and *max* can be justified.

If the propositions are related to different universes, a logical connective result in a multivariable fuzzy set. Consider the following proposition:

$$P : x_1 \text{ is } A_1 \quad \textbf{and } x_2 \text{ is } A_2$$

where A_1 and A_2 have membership functions $\mu_{A_1}(x_1)$ and $\mu_{A_2}(x_2)$. The proposition p can then be represented by a fuzzy set P with the membership function:

$$\mu_P(x_1, x_2) = \mathrm{T}(\mu_{A_1}(x_1), \mu_{A_2}(x_2)), \tag{3.31}$$

where T is a t-norm which models the *and* connective. A combination of propositions is again a proposition.

Negation within a fuzzy proposition is related to the complement of a fuzzy set. For a proposition

$$P : x \text{ is } \quad \textbf{not } A$$

the standard complement results in:

$$\mu_P(x) = 1 - \mu_A(x)$$

Most common is the *conjunctive form* of the antecedent, which is given by:

$$\begin{aligned} \mathcal{R}_i\text{:} \qquad & \textbf{If } x_1 \text{ is } A_{i1} \textbf{ and } x_2 \text{ is } A_{i2} \textbf{ and } \ldots \textbf{ and } x_p \text{ is } A_{ip} \textbf{ then } y \text{ is } B_i, \\ & i = 1, 2, \ldots, K\,. \end{aligned} \tag{3.32}$$

Note that the above model is a special case of (3.1), as the fuzzy set A_i in (3.1) is obtained as the Cartesian product conjunction of fuzzy sets A_{ij}: $A_i = A_{i1} \times A_{i2} \times \cdots \times A_{ip}$. Hence, for a crisp input, the degree of fulfillment (step 1 of Algorithm 3.1) is given by:

$$\beta_i = \mu_{A_{i1}}(x_1) \wedge \mu_{A_{i2}}(x_2) \wedge \cdots \wedge \mu_{A_{ip}}(x_p), \quad 1 \le i \le K\,. \tag{3.33}$$

A set of rules in the conjunctive antecedent form divides the input domain into a lattice of fuzzy hyperboxes, parallel with the axes. Each of the hyperboxes is an Cartesian product-space intersection of the corresponding univariate fuzzy sets. This is shown in Figure 3.12a. The number of rules in the conjunctive form, needed to cover the entire domain, is given by:

$$K = \Pi_{i=1}^{p} N_i,$$

where p is the dimension of the input space and N_i is the number of linguistic terms of the ith antecedent variable.

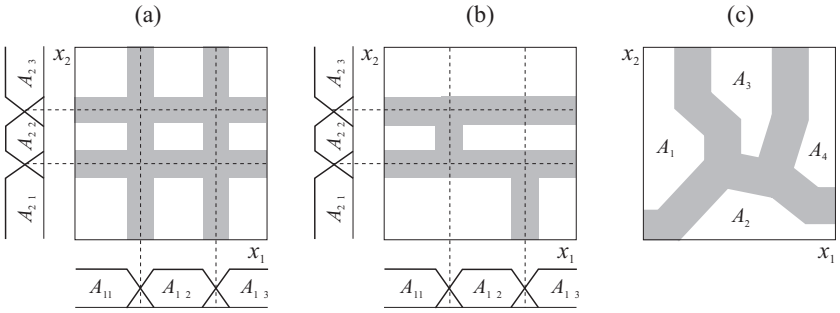


Figure 3.12.: Different partitions of the antecedent space. Gray areas denote the overlapping regions of the fuzzy sets.

By combining conjunctions, disjunctions and negations, various partitions of the antecedent space can be obtained, the boundaries are, however, restricted to the rectangular grid defined by the fuzzy sets of the individual variables, see Figure 3.12b. As an example consider the rule antecedent covering the lower left corner of the antecedent space in this figure:

$$\textbf{If } x_1 \text{ is } \textbf{not } A_{13} \textbf{ and } x_2 \text{ is } A_{21} \textbf{ then } \ldots$$

The degree of fulfillment of this rule is computed using the complement and intersection operators:

$$\beta = [1 - \mu_{A_{13}}(x_1)] \wedge \mu_{A_{21}}(x_2)\,. \tag{3.34}$$

The antecedent form with multivariate membership functions (3.1) is the most general one, as there is no restriction on the shape of the fuzzy regions. The boundaries between these regions can be arbitrarily curved and oblique to the axes, as depicted in Figure 3.12c. Also the number of fuzzy sets needed to cover the antecedent space may be much smaller than in the previous cases. Hence, for complex multivariable systems, this partition may provide the most effective representation. Note that the fuzzy sets A_1 to A_4 in Figure 3.12c still can be projected onto X_1 and X_2 to obtain an approximate linguistic interpretation of the regions described.

3.2.7. Rule Chaining

So far, only a one-layer structure of a fuzzy model has been considered. In practice, however, an output of one rule base may serve as an input to another rule base. This results in a structure with several layers and chained rules. This situation occurs, for instance, in hierarchical models or controller which include several rule bases. Hierarchical organization of knowledge is often used as a natural approach to complexity reduction. A large rule base with many input variables may be split into several interconnected rule bases with fewer inputs. As an example, suppose a rule base with three inputs, each with five linguistic terms. Using the conjunctive form (3.32), 125 rules have to be defined to cover all the input situations. Splitting the rule base in two smaller rule bases, as depicted in Figure 3.13, results in a total of 50 rules.

Another example of rule chaining is the simulation of dynamic fuzzy systems, where a cascade connection of rule bases results from the fact that a value predicted by the model at

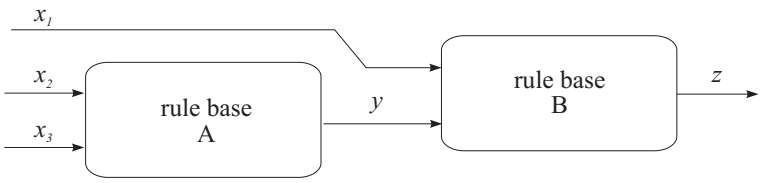


Figure 3.13.: Cascade connection of two rule bases.

time k is used as an input at time $k+1$. As an example, consider a nonlinear discrete-time model

$$\hat{x}(k+1) = f\big(\hat{x}(k), u(k)\big), \tag{3.35}$$

where f is a mapping realized by the rule base, $\hat{x}(k)$ is a predicted state of the process at time k (at the same time it is the state of the model), and $u(k)$ is an input. At the next time step we have:

$$\hat{x}(k+2) = f\big(\hat{x}(k+1), u(k+1)\big) = f\Big(f\big(\hat{x}(k), u(k)\big), u(k+1)\Big), \tag{3.36}$$

which gives a cascade chain of rules.

The hierarchical structure of the rule bases shown in Figure 3.13 requires that the information inferred in Rule base A is passed to Rule base B. This can be accomplished by *defuzzification* at the output of the first rule base and subsequent *fuzzification* at the input of the second rule base. A drawback of this approach is that membership functions have to be defined for the intermediate variable and that a suitable defuzzification method must be chosen. If the values of the intermediate variable cannot be verified by using data, there is no direct way of checking whether the choice is appropriate or not. Also, the fuzziness of the output of the first stage is removed by defuzzification and subsequent fuzzification. This method is used mainly for the simulation of dynamic systems, such as (3.36), when the intermediate variable serves at the same time as a crisp output of the system.

Another possibility is to feed the *fuzzy set* at the output of the first rule base directly (without defuzzification) to the second rule base. An advantage of this approach is that it does not require any additional information from the user. However, in general, the relational composition must be carried out, which requires discretization of the domains and a more complicated implementation. In the case of the Mamdani max-min inference method, the reasoning can be simplified, since the membership degrees of the output fuzzy set directly become the membership degrees of the antecedent propositions where the particular linguistic terms occur. Assume, for instance, that inference in Rule base A results in the following aggregated degrees of fulfillment of the consequent linguistic terms B_1 to B_5 :

$$\boldsymbol{\omega} = [0/B_1,\ 0.7/B_2,\ 0.1/B_3,\ 0/B_4,\ 0/B_5].$$

The membership degree of the propositions "If y is B_2" in rule base B is thus 0.7, the membership degree of the propositions "If y is B_3" is 0.1, and the propositions with the remaining linguistic terms have the membership degree equal to zero.

3.3. Singleton Model

A special case of the linguistic fuzzy model is obtained when the consequent fuzzy sets B_i are singleton sets. These sets can be represented as real numbers b_i, yielding the following rules:

$$\mathcal{R}_i: \quad \textbf{If } \mathbf{x} \text{ is } A_i \textbf{ then } y = b_i, \quad i = 1, 2, \ldots, K\,. \tag{3.37}$$

This model is called the *singleton model*. Contrary to the linguistic model, the number of distinct singletons in the rule base is usually not limited, i.e., each rule may have its own singleton consequent. For the singleton model, the COG defuzzification results in the *fuzzy-mean method*:

$$y = \frac{\sum_{i=1}^{K} \beta_i\, b_i}{\sum_{i=1}^{K} \beta_i}\,. \tag{3.38}$$

Note that here all the K rules contribute to the defuzzification, as opposed to the method given by eq. (3.27). This means that if two rules which have the same consequent singleton are active, this singleton counts twice in the weighted mean (3.38). When using (3.27), each consequent would count only once with a weight equal to the larger of the two degrees of fulfillment. Note that the singleton model can also be seen as a special case of the Takagi–Sugeno model, presented in Section 3.5.

An advantage of the singleton model over the linguistic model is that the consequent parameters b_i can easily be estimated from data, using least-squares techniques. The singleton fuzzy model belongs to a general class of general function approximators, called the *basis functions expansion*, (Friedman, 1991) taking the form:

$$y = \sum_{i=1}^{K} \phi_i(\mathbf{x}) b_i\,. \tag{3.39}$$

Most structures used in nonlinear system identification, such as artificial neural networks, radial basis function networks, or splines, belong to this class of systems. In the singleton model, the basis functions $\phi_i(\mathbf{x})$ are given by the (normalized) degrees of fulfillment of the rule antecedents, and the constants b_i are the consequents. Multilinear interpolation between the rule consequents is obtained if:

- the antecedent membership functions are trapezoidal, pairwise overlapping and the membership degrees sum up to one for each domain element,
- the product operator is used to represent the logical **and** connective in the rule antecedents.

A univariate example is shown in Figure 3.14a.

Clearly, a singleton model can also represent any given linear mapping of the form:

$$y = \mathbf{k}^T\mathbf{x} + q = \sum_{i=1}^{p} k_i x_i + q\,. \tag{3.40}$$

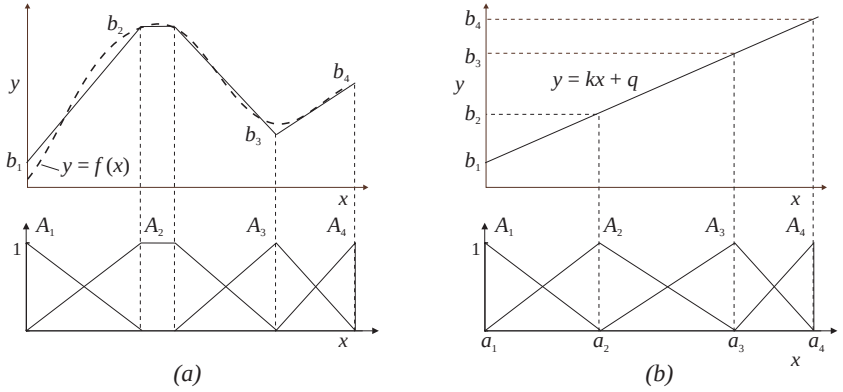


Figure 3.14.: Singleton model with triangular or trapezoidal membership functions results in a piecewise linear input-output mapping (a), of which a linear mapping is a special case (b).

In this case, the antecedent membership functions must be triangular. The consequent singletons can be computed by evaluating the desired mapping (3.40) for the cores a_{ij} of the antecedent fuzzy sets A_{ij} (see Figure 3.14b):

$$b_i = \sum_{j=1}^{p} k_j a_{ij} + q\,. \tag{3.41}$$

This property is useful, as the (singleton) fuzzy model can always be initialized such that it mimics a given (perhaps inaccurate) linear model or controller and can later be optimized.

3.4. Relational Model

Fuzzy relational models (Pedrycz, 1985, 1993) encode associations between linguistic terms defined in the system's input and output domains by using fuzzy relations. The individual elements of the relation represent the strength of association between the fuzzy sets. Let us first consider the already known linguistic fuzzy model which consists of the following rules:

$$\mathcal{R}_i:\ \textbf{If } x_1 \text{ is } A_{i,1} \textbf{ and } \ldots \textbf{ and } x_n \text{ is } A_{i,n} \textbf{ then } y \text{ is } B_i, \quad i = 1, 2, \ldots, K\,. \tag{3.42}$$

Denote $\mathcal{A}_j$ the set of linguistic terms defined for an antecedent variable x_j:

$$\mathcal{A}_j = \{A_{j,l} \mid l = 1, 2, \ldots, N_j\}, \quad j = 1, 2, \ldots, n,$$

where $\mu_{A_{j,l}}(x_j)\colon X_j \to [0, 1]$. Similarly, the set of linguistic terms defined for the consequent variable y is denoted by:

$$\mathcal{B} = \{B_l \mid l = 1, 2, \ldots, M\},$$

with $\mu_{B_l}(y)\colon Y \to [0, 1]$. The key point in understanding the principle of fuzzy relational models is to realize that the rule base (3.42) can be represented as a *crisp* relation S between the antecedent term sets $\mathcal{A}_j$ and the consequent term sets $\mathcal{B}$:

$$S\colon \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n \times \mathcal{B} \to \{0, 1\}\,. \tag{3.43}$$

By denoting $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$ the Cartesian space of the antecedent linguistic terms, (3.43) can be simplified to $S\colon \mathcal{A} \times \mathcal{B} \to \{0, 1\}$. Note that if rules are defined for all possible combinations of the antecedent terms, $K = \mathrm{card}(\mathcal{A})$. Now S can be represented as a $K \times M$ matrix, constrained to only one nonzero element in each row.

Example 3.8 (Relational Representation of a Rule Base) Consider a fuzzy model with two inputs, x_1, x_2, and one output, y. Define two linguistic terms for each input: $\mathcal{A}_1 = \{\mathit{Low}, \mathit{High}\}$, $\mathcal{A}_2 = \{\mathit{Low}, \mathit{High}\}$, and three terms for the output: $\mathcal{B} = \{\mathit{Slow}, \mathit{Moderate}, \mathit{Fast}\}$. For all possible combinations of the antecedent terms, four rules are obtained (the consequents are selected arbitrarily):

If x_1 is *Low* **and** x_2 is *Low* **then** y is *Slow*
If x_1 is *Low* **and** x_2 is *High* **then** y is *Moderate*
If x_1 is *High* **and** x_2 is *Low* **then** y is *Moderate*
If x_1 is *High* **and** x_2 is *High* **then** y is *Fast*.

In this example, $\mathcal{A} = \{(\mathit{Low}, \mathit{Low}), (\mathit{Low}, \mathit{High}), (\mathit{High}, \mathit{Low}), (\mathit{High}, \mathit{High})\}$. The above rule base can be represented by the following relational matrix S:

			y	
x_1	x_2	*Slow*	*Moderate*	*Fast*
Low	*Low*	1	0	0
Low	*High*	0	1	0
High	*Low*	0	1	0
High	*High*	0	0	1

□

The fuzzy relational model is nothing else than an extension of the above crisp relation S to a fuzzy relation $R = [r_{i,j}]$:

$$R\colon \mathcal{A} \times \mathcal{B} \to [0, 1]\,. \tag{3.44}$$

Each rule now contains all the possible consequent terms, each with its own weighting factor, given by the respective element r_{ij} of the fuzzy relation (Figure 3.15). This weighting allows the model to be fine-tuned more easily for instance to fit data.

It should be stressed that the relation R in (3.44) is different from the relation (3.18) encoding linguistic if–then rules. The latter relation is a multidimensional membership function defined in the product space of the input and output domains, whose each element represents the degree of association between the individual *crisp* elements in the antecedent and consequent domains. In fuzzy relational models, however, the relation represents associations between the individual *linguistic terms*.

Example 3.9 (Relational model) Using the linguistic terms of Example 3.8, a fuzzy relational model can be defined, for instance, by the following relation R:

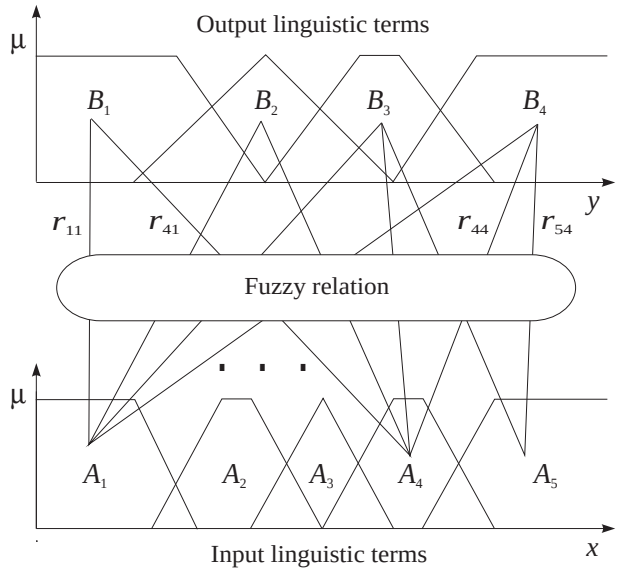


Figure 3.15.: Fuzzy relation as a mapping from input to output linguistic terms.

			y	
x_1	x_2	*Slow*	*Moderate*	*Fast*
Low	*Low*	0.9	0.2	0.0
Low	*High*	0.0	1.0	0.0
High	*Low*	0.0	0.8	0.2
High	*High*	0.0	0.1	0.8

The elements $r_{i,j}$ describe the associations between the combinations of the antecedent linguistic terms and the consequent linguistic terms. This implies that the consequents are not exactly equal to the predefined linguistic terms, but are given by their weighted combinations. Note that the sum of the weights does not have to equal one. In terms of rules, this relation can be interpreted as:

If x_1 is *Low* **and** x_2 is *Low* **then** y is *Slow* (0.9), y is *Mod.* (0.2), y is *Fast* (0.0)
If x_1 is *Low* **and** x_2 is *High* **then** y is *Slow* (0.0), y is *Mod.* (1.0), y is *Fast* (0.0)
If x_1 is *High* **and** x_2 is *Low* **then** y is *Slow* (0.0), y is *Mod.* (0.8), y is *Fast* (0.2)
If x_1 is *High* **and** x_2 is *High* **then** y is *Slow* (0.0), y is *Mod.* (0.1), y is *Fast* (0.8).

The numbers in parentheses are the respective elements $r_{i,j}$ of R.

□

The inference is based on the relational composition (2.42) of the fuzzy set representing the degrees of fulfillment β_i and the relation R. It is given in the following algorithm.

Algorithm 3.2 *Inference in fuzzy relational model.*

1. Compute the degree of fulfillment by:

$$\beta_i = \mu_{A_{i1}}(x_1) \wedge \cdots \wedge \mu_{A_{ip}}(x_p),\ i = 1, 2, \ldots, K\,. \tag{3.45}$$

(Other intersection operators, such as the product, can be applied as well.)

2. Apply the relational composition $\boldsymbol{\omega} = \boldsymbol{\beta} \circ R$, given by:

$$\omega_j = \max_{1 \le i \le K} \big(\beta_i \wedge r_{ij}\big), \quad j = 1, 2, \ldots, M\,. \tag{3.46}$$

3. Defuzzify the consequent fuzzy set by:

$$y = \frac{\sum_{l=1}^{M} \omega_l \cdot b_l}{\sum_{l=1}^{M} \omega_l} \tag{3.47}$$

where b_l are the centroids of the consequent fuzzy sets B_l computed by applying some defuzzification method such as the center-of-gravity (3.25) or the mean-of-maxima (3.26) to the individual fuzzy sets B_l.

Note that if R is crisp, the Mamdani inference with the fuzzy-mean defuzzification (3.27) is obtained.

Example 3.10 (Inference) Suppose, that for the rule base of Example 3.8, the following membership degrees are obtained:

$$\mu_{Low}(x_1) = 0.9, \quad \mu_{High}(x_1) = 0.2, \quad \mu_{Low}(x_2) = 0.6, \quad \mu_{High}(x_2) = 0.3,$$

for some given inputs x_1 and x_2. To infer y, first apply (3.45) to obtain $\boldsymbol{\beta}$. Using the product t-norm, the following values are obtained:

$$\beta_1 = \mu_{Low}(x_1) \cdot \mu_{Low}(x_2) = 0.54 \qquad \beta_2 = \mu_{Low}(x_1) \cdot \mu_{High}(x_2) = 0.27$$
$$\beta_3 = \mu_{High}(x_1) \cdot \mu_{Low}(x_2) = 0.12 \qquad \beta_4 = \mu_{High}(x_1) \cdot \mu_{High}(x_2) = 0.06$$

Hence, the degree of fulfillment is: $\boldsymbol{\beta} = [0.54,\ 0.27,\ 0.12,\ 0.06]$. Now we apply (3.46) to obtain the output fuzzy set $\boldsymbol{\omega}$:

$$\boldsymbol{\omega} = \boldsymbol{\beta} \circ R = [0.54,\ 0.27,\ 0.12,\ 0.06] \circ \begin{bmatrix} 0.9 & 0.2 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.8 & 0.2 \\ 0.0 & 0.1 & 0.8 \end{bmatrix} = [0.54,\ 0.27,\ 0.12]\,. \tag{3.48}$$

Finally, by using (3.47), the defuzzified output y is computed:

$$y = \frac{0.54\,\mathrm{cog}(\mathit{Slow}) + 0.27\,\mathrm{cog}(\mathit{Moderate}) + 0.12\,\mathrm{cog}(\mathit{Fast})}{0.54 + 0.27 + 0.12}\,. \tag{3.49}$$

□

The main advantage of the relational model is that the input–output mapping can be fine-tuned without changing the consequent fuzzy sets (linguistic terms). In the linguistic model, the outcomes of the individual rules are restricted to the grid given by the centroids of the output fuzzy sets, which is not the case in the relational model, see Figure 3.16.

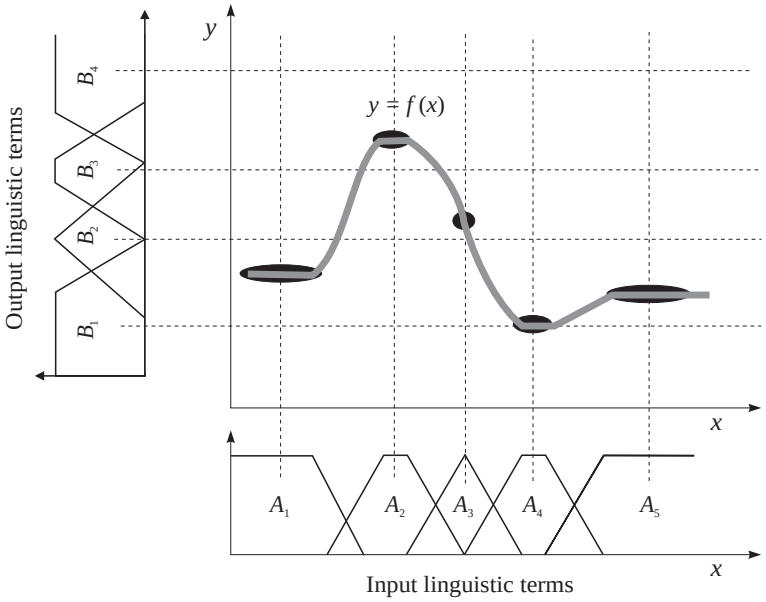


Figure 3.16.: A possible input–output mapping of a fuzzy relational model.

For this additional degree of freedom, one pays by having more free parameters (elements in the relation). If no constraints are imposed on these parameters, several elements in a row of R can be nonzero, which may hamper the interpretation of the model. Furthermore, the shape of the output fuzzy sets has no influence on the resulting defuzzified value, since only centroids of these sets are considered in defuzzification.

It is easy to verify that if the antecedent fuzzy sets sum up to one and the bounded-sum–product composition is used, a relational model can be computationally replaced by an equivalent model with singleton consequents (Voisin et al., 1995).

Example 3.11 (Relational and Singleton Model) Fuzzy relational model:

$$
\begin{aligned}
&\textbf{If } x \text{ is } A_1 \textbf{ then} && y \text{ is } B_1\ (0.8),\ \ y \text{ is } B_2\ (0.1),\ \ y \text{ is } B_3\ (0.0)\,.\\
&\textbf{If } x \text{ is } A_2 \textbf{ then} && y \text{ is } B_1\ (0.6),\ \ y \text{ is } B_2\ (0.2),\ \ y \text{ is } B_3\ (0.0)\,.\\
&\textbf{If } x \text{ is } A_3 \textbf{ then} && y \text{ is } B_1\ (0.5),\ \ y \text{ is } B_2\ (0.7),\ \ y \text{ is } B_3\ (0.0)\,.\\
&\textbf{If } x \text{ is } A_4 \textbf{ then} && y \text{ is } B_1\ (0.0),\ \ y \text{ is } B_2\ (0.1),\ \ y \text{ is } B_3\ (0.9),
\end{aligned}
$$

can be replaced by the following singleton model:

$$
\begin{aligned}
&\textbf{If } x \text{ is } A_1 \textbf{ then} && y = (0.8b_1 + 0.1b_2)/(0.8 + 0.1),\\
&\textbf{If } x \text{ is } A_2 \textbf{ then} && y = (0.6b_1 + 0.2b_2)/(0.6 + 0.2),\\
&\textbf{If } x \text{ is } A_3 \textbf{ then} && y = (0.5b_1 + 0.7b_2)/(0.5 + 0.7),\\
&\textbf{If } x \text{ is } A_4 \textbf{ then} && y = (0.1b_2 + 0.9b_3)/(0.1 + 0.9),
\end{aligned}
$$

where b_j are defuzzified values of the fuzzy sets B_j, $b_j = \mathrm{cog}(B_j)$.

□

If also the consequent membership functions form a partition, a singleton model can conversely be expressed as an equivalent relational model by computing the membership degrees of the singletons in the consequent fuzzy sets B_j. These membership degrees then become the elements of the fuzzy relation:

$$R = \begin{bmatrix} \mu_{B_1}(b_1) & \mu_{B_2}(b_1) & \dots & \mu_{B_M}(b_1) \\ \mu_{B_1}(b_2) & \mu_{B_2}(b_2) & \dots & \mu_{B_M}(b_2) \\ \vdots & \vdots & \vdots & \vdots \\ \mu_{B_1}(b_K) & \mu_{B_2}(b_K) & \dots & \mu_{B_M}(b_K) \end{bmatrix} . \tag{3.50}$$

Clearly, the linguistic model is a special case of the fuzzy relational model, with R being a crisp relation constrained such that only one nonzero element is allowed in each row of R (each rule has only one consequent).

3.5. Takagi–Sugeno Model

The Takagi–Sugeno (TS) fuzzy model (Takagi and Sugeno, 1985), on the other hand, uses crisp functions in the consequents. Hence, it can be seen as a combination of linguistic and mathematical regression modeling in the sense that the antecedents describe fuzzy regions in the input space in which the consequent functions are valid. The TS rules have the following form:

$$\mathcal{R}_i\colon \ \textbf{If } \mathbf{x} \text{ is } A_i \textbf{ then } \mathbf{y}_i = \mathbf{f}_i(\mathbf{x}), \qquad i = 1, 2, \dots, K\,. \tag{3.51}$$

Contrary to the linguistic model, the input $\mathbf{x}$ is a crisp variable (linguistic inputs are in principle possible, but would require the use of the extension principle (Zadeh, 1975) to compute the fuzzy value of $\mathbf{y}_i$). The functions $\mathbf{f}_i$ are typically of the same structure, only the parameters in each rule are different. Generally, $\mathbf{f}_i$ is a vector-valued function, but for the ease of notation we will consider a scalar f_i in the sequel. A simple and practically useful parameterization is the affine (linear in parameters) form, yielding the rules:

$$\mathcal{R}_i\colon \ \textbf{If } \mathbf{x} \text{ is } A_i \textbf{ then } y_i = \mathbf{a}_i^T\mathbf{x} + b_i, \qquad i = 1, 2, \dots, K, \tag{3.52}$$

where $\mathbf{a}_i$ is a parameter vector and b_i is a scalar offset. This model is called an *affine TS model*. Note that if $\mathbf{a}_i = 0$ for each i, the singleton model (3.37) is obtained.

3.5.1. Inference in the TS Model

The inference formula of the TS model is a straightforward extension of the singleton model inference (3.38):

$$y = \frac{\sum_{i=1}^{K} \beta_i y_i}{\sum_{i=1}^{K} \beta_i} = \frac{\sum_{i=1}^{K} \beta_i (\mathbf{a}_i^T\mathbf{x} + b_i)}{\sum_{i=1}^{K} \beta_i} . \tag{3.53}$$

When the antecedent fuzzy sets define distinct but overlapping regions in the antecedent space and the parameters $\mathbf{a}_i$ and b_i correspond to a local linearization of a nonlinear function, the TS model can be regarded as a smoothed piece-wise approximation of that function, see Figure 3.17.

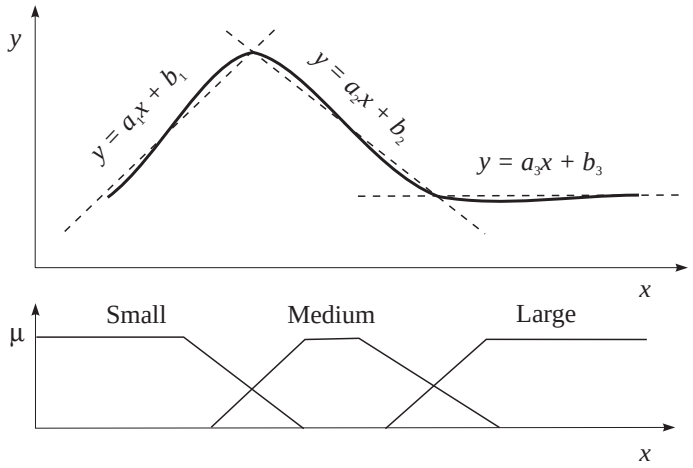


Figure 3.17.: Takagi–Sugeno fuzzy model as a smoothed piece-wise linear approximation of a nonlinear function.

3.5.2. TS Model as a Quasi-Linear System

The affine TS model can be regarded as a quasi-linear system (i.e., a linear system with input-dependent parameters). To see this, denote the normalized degree of fulfillment by

$$\gamma_i(\mathbf{x}) = \frac{\beta_i(\mathbf{x})}{\sum_{j=1}^{K} \beta_j(\mathbf{x})} \,. \tag{3.54}$$

Here we write $\beta_i(\mathbf{x})$ explicitly as a function $\mathbf{x}$ to stress that the TS model is a quasi-linear model of the following form:

$$y = \left(\sum_{i=1}^{K} \gamma_i(\mathbf{x}) \mathbf{a}_i^T \right) \mathbf{x} + \sum_{i=1}^{K} \gamma_i(\mathbf{x}) b_i = \mathbf{a}^T(\mathbf{x})\mathbf{x} + b(\mathbf{x}) \,. \tag{3.55}$$

The 'parameters' $\mathbf{a}(\mathbf{x})$, $b(\mathbf{x})$ are convex linear combinations of the consequent parameters $\mathbf{a}_i$ and b_i, i.e.:

$$\mathbf{a}(\mathbf{x}) = \sum_{i=1}^{K} \gamma_i(\mathbf{x}) \mathbf{a}_i, \quad b(\mathbf{x}) = \sum_{i=1}^{K} \gamma_i(\mathbf{x}) b_i \,. \tag{3.56}$$

In this sense, a TS model can be regarded as a mapping from the antecedent (input) space to a convex region (polytope) in the space of the parameters of a quasi-linear system, as schematically depicted in Figure 3.18.

This property facilitates the analysis of TS models in a framework similar to that of linear systems. Methods have been developed to design controllers with desired closed loop characteristics (Filev, 1996) and to analyze their stability (Tanaka and Sugeno, 1992; Zhao, 1995; Tanaka et al., 1996).

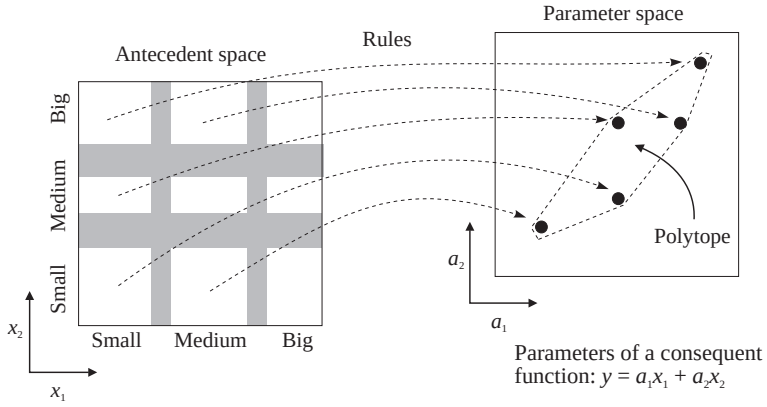


Figure 3.18.: A TS model with affine consequents can be regarded as a mapping from the antecedent space to the space of the consequent parameters.

3.6. Dynamic Fuzzy Systems

In system modeling and identification one often deals with the approximation of dynamic systems. Time-invariant dynamic systems are in general modeled as static functions, by using the concept of the system's *state*. Given the state of a system and given its input, we can determine what the next state will be. In the discrete-time setting we can write

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k)), \tag{3.57}$$

where $\mathbf{x}(k)$ and $\mathbf{u}(k)$ are the state and the input at time k, respectively, and $\mathbf{f}$ is a static function, called the *state-transition function*. Fuzzy models of different types can be used to approximate the state-transition function. As the state of a process is often not measured, *input-output* modeling is often applied. The most common is the NARX (Nonlinear AutoRegressive with eXogenous input) model:

$$y(k+1) = f(y(k), y(k-1), \ldots, y(k-n_y+1), u(k), u(k-1), \ldots, u(k-n_u+1)) \,. \tag{3.58}$$

Here $y(k), \ldots, y(k-n_y+1)$ and $u(k), \ldots, u(k-n_u+1)$ denote the past model outputs and inputs respectively and n_y, n_u are integers related to the order of the dynamic system. For example, a singleton fuzzy model of a dynamic system may consist of rules of the following form:

$$\begin{aligned} \mathcal{R}_i\colon \quad & \textbf{If } y(k) \text{ is } A_{i1} \textbf{ and } y(k-1) \text{ is } A_{i2} \textbf{ and} \ldots y(k-n+1) \text{ is } A_{in} \\ & \textbf{and } u(k) \text{ is } B_{i1} \textbf{ and } u(k-1) \text{ is } B_{i2} \textbf{ and} \ldots u(k-m+1) \text{ is } B_{im} \\ & \textbf{then } y(k+1) \text{ is } c_i \,. \end{aligned} \tag{3.59}$$

In this sense, we can say that the dynamic behavior is taken care of by external dynamic filters added to the fuzzy system, see Figure 3.19. In (3.59), the input dynamic filter is a simple generator of the lagged inputs and outputs, and no output filter is used.

A dynamic TS model is a sort of parameter-scheduling system. It has in its consequents

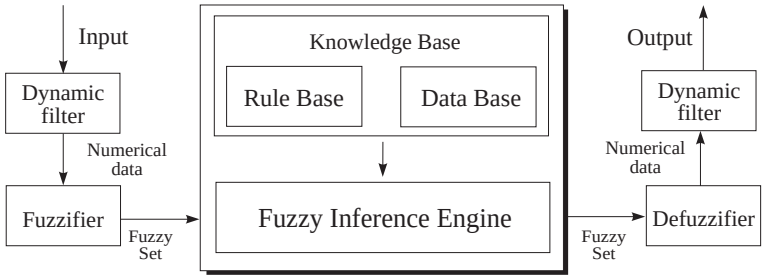


Figure 3.19.: A generic fuzzy system with fuzzification and defuzzification units and external dynamic filters.

linear ARX models whose parameters are generally different in each rule:

$$\begin{aligned}\mathcal{R}_i: \quad & \textbf{If } y(k) \text{ is } A_{i1} \textbf{ and } y(k-1) \text{ is } A_{i2} \textbf{ and } \ldots\ y(k-n_y+1) \text{ is } A_{in_y} \\ & \textbf{and } u(k) \text{ is } B_{i1} \textbf{ and } u(k-1) \text{ is } B_{i2} \textbf{ and } \ldots\ u(k-n_u+1) \text{ is } B_{in_u} \\ & \textbf{then } y(k+1) = \sum_{j=1}^{n_y} a_{ij} y(k-j+1) + \sum_{j=1}^{n_u} b_{ij} u(k-j+1) + c_i\,. \end{aligned} \tag{3.60}$$

Besides these frequently used input–output systems, fuzzy models can also represent nonlinear systems in the state-space form:

$$\begin{aligned}\mathbf{x}(k+1) &= g(\mathbf{x}(k), \mathbf{u}(k)) \\ \mathbf{y}(k) &= h(\mathbf{x}(k))\end{aligned}$$

where state transition function g maps the current state $\mathbf{x}(k)$ and the input $\mathbf{u}(k)$ into a new state $\mathbf{x}(k+1)$. The output function h maps the state $\mathbf{x}(k)$ into the output $\mathbf{y}(k)$. An example of a rule-based representation of a state-space model is the following Takagi–Sugeno model:

$$\textbf{If } \mathbf{x}(k) \text{ is } A_i \textbf{ and } \mathbf{u}(k) \text{ is } B_i \textbf{ then } \begin{cases} \mathbf{x}(k+1) = \mathbf{A}_i\mathbf{x}(k) + \mathbf{B}_i\mathbf{u}(k) + \mathbf{a}_i \\ \mathbf{y}(k) \quad\ = \mathbf{C}_i\mathbf{x}(k) + \mathbf{c}_i \end{cases} \tag{3.61}$$

for $i = 1, \ldots, K$. Here $\mathbf{A}_i$, $\mathbf{B}_i$, $\mathbf{C}_i$, $\mathbf{a}_i$ and $\mathbf{c}_i$ are matrices and vectors of appropriate dimensions, associated with the ith rule.

The state-space representation is useful when the prior knowledge allows us to model the system from first principles such as mass and energy balances. In literature, this approach is called white-box state-space modeling (Ljung, 1987). If the state is directly measured on the system, or can be reconstructed from other measured variables, both g and h can be approximated by using nonlinear regression techniques. An advantage of the state-space modeling approach is that the structure of the model is related to the structure of the real system, and, consequently, also the model parameters are often physically relevant. This is usually not the case in the input-output models. In addition, the dimension of the regression problem in state-space modeling is often smaller than with input–output models, since the state of the system can be represented with a vector of a lower dimension than the regression (3.58).

Since fuzzy models are able to approximate any smooth function to any degree of accuracy, (Wang, 1992) models of type (3.59), (3.60) and (3.61) can approximate any observable and controllable modes of a large class of discrete-time nonlinear systems (Leonaritis and Billings, 1985).

3.7. Summary and Concluding Remarks

This chapter has reviewed four different types of rule-based fuzzy models: linguistic (Mamdani-type) models, fuzzy relational models, singleton and Takagi–Sugeno models. A major distinction can be made between the linguistic model, which has fuzzy sets in both the rule antecedents and consequents of the rules, and the TS model, where the consequents are (crisp) functions of the input variables. Fuzzy relational models can be regarded as an extension of linguistic models, which allow for different degrees of association between the antecedent and the consequent linguistic terms.

3.8. Problems

1. Give a definition of a linguistic variable. What is the difference between linguistic variables and linguistic terms?

2. The minimum t-norm can be used to represent if-then rules in a similar way as fuzzy implications. It is however not an implication function. Explain why. Give at least one example of a function that is a proper fuzzy implication.

3. Consider a rule **If** x is A **then** y is B with fuzzy sets $A = \{0.1/x_1,\ 0.4/x_2,\ 1/x_3\}$ and $B = \{0/y_1,\ 1/y_2,\ 0.2/y_3\}$. Compute the fuzzy relation R that represents the truth value of this fuzzy rule. Use first the minimum t-norm and then the Łukasiewicz implication. Discuss the difference in the results.

4. Explain the steps of the Mamdani (max-min) inference algorithm for a linguistic fuzzy system with one (crisp) input and one (fuzzy) output. Apply these steps to the following rule base:

   1) **If** x is A_1 **then** y is B_1,
   2) **If** x is A_2 **then** y is B_2,

   with

   $$A_1 = \{0.1/1,\ 0.6/2,\ 1/3\}, \quad A_2 = \{0.9/1,\ 0.4/2,\ 0/3\},$$
   $$B_1 = \{1/4,\ 1/5,\ 0.3/6\}, \quad B_2 = \{0.1/4,\ 0.9/5,\ 1/6\},$$

   State the inference in terms of equations. Compute the output fuzzy set B' for $x = 2$.

5. Define the center-of-gravity and the mean-of-maxima defuzzification methods. Apply them to the fuzzy set $B = \{0.1/1,\ 0.2/2,\ 0.7/3,\ 1/4\}$ and compare the numerical results.

6. Consider the following Takagi–Sugeno rules:

   1) **If** x is A_1 **and** y is B_1 **then** $z_1 = x + y + 1$
   2) **If** x is A_2 **and** y is B_1 **then** $z_2 = 2x + y + 1$
   3) **If** x is A_1 **and** y is B_2 **then** $z_3 = 2x + 3y$
   4) **If** x is A_2 **and** y is B_2 **then** $z_4 = 2x + 5$

Give the formula to compute the output z and compute the value of z for $x = 1$, $y = 4$ and the antecedent fuzzy sets

$$\begin{array}{rclcrcl} A_1 & = & \{0.1/1,\ 0.6/2,\ 1/3\}, & & A_2 & = & \{0.9/1,\ 0.4/2,\ 0/3\}, \\ B_1 & = & \{1/4,\ 1/5,\ 0.3/6\}, & & B_2 & = & \{0.1/4,\ 0.9/5,\ 1/6\}\,. \end{array}$$

7. Consider an unknown dynamic system $y(k+1) = f\big(y(k), u(k)\big)$. Give an example of a singleton fuzzy model that can be used to approximate this system. What are the free parameters in this model?

4. Fuzzy Clustering

Clustering techniques are mostly unsupervised methods that can be used to organize data into groups based on similarities among the individual data items. Most clustering algorithms do not rely on assumptions common to conventional statistical methods, such as the underlying statistical distribution of data, and therefore they are useful in situations where little prior knowledge exists. The potential of clustering algorithms to reveal the underlying structures in data can be exploited in a wide variety of applications, including classification, image processing, pattern recognition, modeling and identification.

This chapter presents an overview of fuzzy clustering algorithms based on the c-means functional. Readers interested in a deeper and more detailed treatment of fuzzy clustering may refer to the classical monographs by Duda and Hart (1973), Bezdek (1981) and Jain and Dubes (1988). A more recent overview of different clustering algorithms can be found in (Bezdek and Pal, 1992).

4.1. Basic Notions

The basic notions of data, clusters and cluster prototypes are established and a broad overview of different clustering approaches is given.

4.1.1. The Data Set

Clustering techniques can be applied to data that are quantitative (numerical), qualitative (categorical), or a mixture of both. In this chapter, the clustering of quantitative data is considered. The data are typically observations of some physical process. Each observation consists of n measured variables, grouped into an n-dimensional column vector $\mathbf{z}_k = [z_{1k}, \ldots, z_{nk}]^T$, $\mathbf{z}_k \in \mathbb{R}^n$. A set of N observations is denoted by $\mathbf{Z} = \{\mathbf{z}_k \mid k = 1, 2, \ldots, N\}$, and is represented as an $n \times N$ matrix:

$$\mathbf{Z} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1N} \\ z_{21} & z_{22} & \cdots & z_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nN} \end{bmatrix}. \tag{4.1}$$

In the pattern-recognition terminology, the columns of this matrix are called *patterns* or objects, the rows are called the *features* or attributes, and $\mathbf{Z}$ is called the *pattern* or *data matrix*. The meaning of the columns and rows of $\mathbf{Z}$ depends on the context. In medical diagnosis, for instance, the columns of $\mathbf{Z}$ may represent patients, and the rows are then symptoms, or laboratory measurements for these patients. When clustering is applied to the modeling and identification of dynamic systems, the columns of $\mathbf{Z}$ may contain samples

of time signals, and the rows are, for instance, physical variables observed in the system (position, pressure, temperature, etc.). In order to represent the system's dynamics, past values of these variables are typically included in $\mathbf{Z}$ as well.

4.1.2. Clusters and Prototypes

Various definitions of a cluster can be formulated, depending on the objective of clustering. Generally, one may accept the view that a cluster is a group of objects that are more similar to one another than to members of other clusters (Bezdek, 1981; Jain and Dubes, 1988). The term "similarity" should be understood as mathematical similarity, measured in some well-defined sense. In metric spaces, similarity is often defined by means of a *distance norm*. Distance can be measured among the data vectors themselves, or as a distance from a data vector to some *prototypical object* (prototype) of the cluster. The prototypes are usually not known beforehand, and are sought by the clustering algorithms simultaneously with the partitioning of the data. The prototypes may be vectors of the same dimension as the data objects, but they can also be defined as "higher-level" geometrical objects, such as linear or nonlinear subspaces or functions.

Data can reveal clusters of different geometrical shapes, sizes and densities as demonstrated in Figure 4.1. While clusters (a) are spherical, clusters (b) to (d) can be characterized as linear and nonlinear subspaces of the data space. The performance of most clustering algorithms is influenced not only by the geometrical shapes and densities of the individual clusters, but also by the spatial relations and distances among the clusters. Clusters can be well-separated, continuously connected to each other, or overlapping each other.

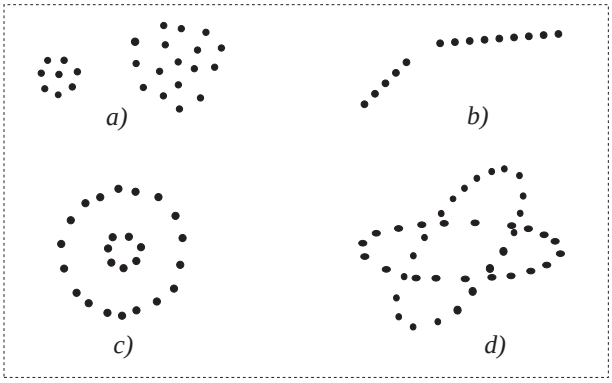


Figure 4.1.: Clusters of different shapes and dimensions in $\mathbb{R}^2$. After (Jain and Dubes, 1988).

4.1.3. Overview of Clustering Methods

Many clustering algorithms have been introduced in the literature. Since clusters can formally be seen as subsets of the data set, one possible classification of clustering methods can be according to whether the subsets are *fuzzy* or *crisp* (hard).

Hard clustering methods are based on classical set theory, and require that an object either does or does not belong to a cluster. Hard clustering means partitioning the data into a specified number of mutually exclusive subsets.

Fuzzy clustering methods, however, allow the objects to belong to several clusters simultaneously, with different degrees of membership. In many situations, fuzzy clustering is more natural than hard clustering. Objects on the boundaries between several classes are not forced to fully belong to one of the classes, but rather are assigned membership degrees between 0 and 1 indicating their partial membership. The discrete nature of the hard partitioning also causes difficulties with algorithms based on analytic functionals, since these functionals are not differentiable.

Another classification can be related to the algorithmic approach of the different techniques (Bezdek, 1981).

- *Agglomerative hierarchical methods* and *splitting hierarchical methods* form new clusters by reallocating memberships of one point at a time, based on some suitable measure of similarity.
- With *graph-theoretic methods*, $\mathbf{Z}$ is regarded as a set of nodes. Edge weights between pairs of nodes are based on a measure of similarity between these nodes.
- Clustering algorithms may use an *objective function* to measure the desirability of partitions. Nonlinear optimization algorithms are used to search for local optima of the objective function.

The remainder of this chapter focuses on fuzzy clustering with objective function. These methods are relatively well understood, and mathematical results are available concerning the convergence properties and cluster validity assessment.

4.2. Hard and Fuzzy Partitions

The concept of *fuzzy partition* is essential for cluster analysis, and consequently also for the identification techniques that are based on fuzzy clustering. Fuzzy and possibilistic partitions can be seen as a generalization of *hard partition* which is formulated in terms of classical subsets.

4.2.1. Hard Partition

The objective of clustering is to partition the data set $\mathbf{Z}$ into c clusters (groups, classes). For the time being, assume that c is known, based on prior knowledge, for instance. Using classical sets, a *hard partition* of $\mathbf{Z}$ can be defined as a family of subsets $\{A_i \mid 1 \leq i \leq c\} \subset \mathcal{P}(\mathbf{Z})$[1] with the following properties (Bezdek, 1981):

$$\bigcup_{i=1}^{c} A_i = \mathbf{Z}, \tag{4.2a}$$

$$A_i \cap A_j = \emptyset, \qquad 1 \leq i \neq j \leq c, \tag{4.2b}$$

$$\emptyset \subset A_i \subset \mathbf{Z}, \qquad 1 \leq i \leq c\,. \tag{4.2c}$$

[1] $\mathcal{P}(Z)$ is the power set of Z.

Equation (4.2a) means that the union subsets A_i contains all the data. The subsets must be disjoint, as stated by (4.2b), and none of them is empty nor contains all the data in $\mathbf{Z}$ (4.2c). In terms of *membership (characteristic) functions*, a partition can be conveniently represented by the *partition matrix* $\mathbf{U} = [\mu_{ik}]_{c\times N}$. The ith row of this matrix contains values of the membership function μ_i of the ith subset A_i of $\mathbf{Z}$. It follows from (4.2) that the elements of $\mathbf{U}$ must satisfy the following conditions:

$$\mu_{ik} \in \{0, 1\}, \qquad 1 \le i \le c, \quad 1 \le k \le N, \tag{4.3a}$$

$$\sum_{i=1}^{c} \mu_{ik} = 1, \qquad 1 \le k \le N, \tag{4.3b}$$

$$0 < \sum_{k=1}^{N} \mu_{ik} < N, \qquad 1 \le i \le c\,. \tag{4.3c}$$

Note that we have "strict" inequalities in (4.3c) as we need to avoid that a single cluster has all the elements belonging to it ($< N$) and to avoid that a cluster has no element at all ($0 <$).

The space of all possible hard partition matrices for $\mathbf{Z}$, called the hard partitioning space (Bezdek, 1981), is thus defined by

$$M_{hc} = \left\{ \mathbf{U} \in \mathbb{R}^{c\times N} \middle| \mu_{ik} \in \{0, 1\}, \forall i, k;\ \sum_{i=1}^{c} \mu_{ik} = 1, \forall k;\ 0 < \sum_{k=1}^{N} \mu_{ik} < N, \forall i \right\}.$$

Example 4.1 ***Hard partition.*** Let us illustrate the concept of hard partition by a simple example. Consider a data set $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_{10}\}$, shown in Figure 4.2.

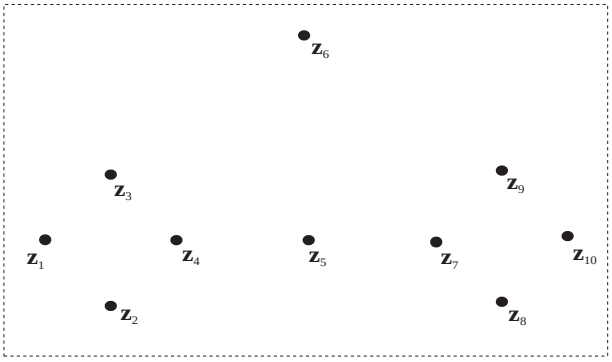


Figure 4.2.: A data set in $\mathbb{R}^2$.

A visual inspection of this data may suggest two well-separated clusters (data points $\mathbf{z}_1$ to $\mathbf{z}_4$ and $\mathbf{z}_7$ to $\mathbf{z}_{10}$ respectively), one point in between the two clusters ($\mathbf{z}_5$), and an "outlier" $\mathbf{z}_6$. One particular partition $\mathbf{U} \in M_{hc}$ of the data into two subsets (out of the 2^{10} possible hard partitions) is

$$\mathbf{U} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The first row of $\mathbf{U}$ defines point-wise the characteristic function for the first subset of $\mathbf{Z}$, A_1, and the second row defines the characteristic function of the second subset of $\mathbf{Z}$, A_2. Each sample must be assigned exclusively to one subset (cluster) of the partition. In this case, both the boundary point $\mathbf{z}_5$ and the outlier $\mathbf{z}_6$ have been assigned to A_1. It is clear that a hard partitioning may not give a realistic picture of the underlying data. Boundary data points may represent patterns with a mixture of properties of data in A_1 and A_2, and therefore cannot be fully assigned to either of these classes, or do they constitute a separate class. This shortcoming can be alleviated by using fuzzy and possibilistic partitions as shown in the following sections.

□

4.2.2. Fuzzy Partition

Generalization of the hard partition to the fuzzy case follows directly by allowing μ_{ik} to attain real values in $[0, 1]$. Conditions for a fuzzy partition matrix, analogous to (4.3) are given by (Ruspini, 1970):

$$\mu_{ik} \in [0, 1], \qquad 1 \leq i \leq c, \quad 1 \leq k \leq N, \tag{4.4a}$$

$$\sum_{i=1}^{c} \mu_{ik} = 1, \qquad 1 \leq k \leq N, \tag{4.4b}$$

$$0 < \sum_{k=1}^{N} \mu_{ik} < N, \qquad 1 \leq i \leq c\,. \tag{4.4c}$$

The ith row of the fuzzy partition matrix $\mathbf{U}$ contains values of the ith *membership function* of the fuzzy subset A_i of $\mathbf{Z}$. Equation (4.4b) constrains the sum of each column to 1, and thus the total membership of each $\mathbf{z}_k$ in $\mathbf{Z}$ equals one. The fuzzy partitioning space for $\mathbf{Z}$ is the set

$$M_{fc} = \left\{ \mathbf{U} \in \mathbb{R}^{c \times N} \middle| \mu_{ik} \in [0, 1], \forall i, k;\ \sum_{i=1}^{c} \mu_{ik} = 1, \forall k;\ 0 < \sum_{k=1}^{N} \mu_{ik} < N, \forall i \right\}.$$

Example 4.2 ***Fuzzy partition.***Consider the data set from Example 4.1. One of the infinitely many fuzzy partitions in $\mathbf{Z}$ is:

$$\mathbf{U} = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 0.8 & 0.5 & 0.5 & 0.2 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.2 & 0.5 & 0.5 & 0.8 & 1.0 & 1.0 & 1.0 \end{bmatrix}.$$

The boundary point $\mathbf{z}_5$ has now a membership degree of 0.5 in both classes, which correctly reflects its position in the middle between the two clusters. Note, however, that the outlier $\mathbf{z}_6$ has the same pair of membership degrees, even though it is further from the two clusters, and thus can be considered less typical of both A_1 and A_2 than $\mathbf{z}_5$. This is because condition (4.4b) requires that the sum of memberships of each point equals one. It can be, of course, argued that three clusters are more appropriate in this example than two. In general, however, it is difficult to detect outliers and assign them to extra clusters. The use of

possibilistic partition, presented in the next section, overcomes this drawback of fuzzy partitions.

□

4.2.3. Possibilistic Partition

A more general form of fuzzy partition, the *possibilistic partition*,[2] can be obtained by relaxing the constraint (4.4b). This constraint, however, cannot be completely removed, in order to ensure that each point is assigned to at least one of the fuzzy subsets with a membership greater than zero. Equation (4.4b) can be replaced by a less restrictive constraint $\forall k,\ \exists i,\ \mu_{ik} > 0$. The conditions for a possibilistic fuzzy partition matrix are:

$$\mu_{ik} \in [0,1], \qquad 1 \leq i \leq c, \quad 1 \leq k \leq N, \tag{4.5a}$$

$$\exists i,\ \mu_{ik} > 0, \qquad \forall k, \tag{4.5b}$$

$$0 < \sum_{k=1}^{N} \mu_{ik} < N, \qquad 1 \leq i \leq c\,. \tag{4.5c}$$

Analogously to the previous cases, the possibilistic partitioning space for $\mathbf{Z}$ is the set

$$M_{pc} = \left\{ \mathbf{U} \in \mathbb{R}^{c \times N} \middle| \mu_{ik} \in [0,1], \forall i,k;\ \forall k,\ \exists i,\ \mu_{ik} > 0;\ 0 < \sum_{k=1}^{N} \mu_{ik} < N, \forall i \right\}.$$

Example 4.3 ***Possibilistic partition.*** An example of a possibilistic partition matrix for our data set is:

$$\mathbf{U} = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 0.5 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.5 & 0.2 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}.$$

As the sum of elements in each column of $\mathbf{U} \in M_{fc}$ is no longer constrained, the outlier has a membership of 0.2 in both clusters, which is lower than the membership of the boundary point $\mathbf{z}_5$, reflecting the fact that this point is less typical for the two clusters than $\mathbf{z}_5$.

□

4.3. Fuzzy c-Means Clustering

Most analytical fuzzy clustering algorithms (and also all the algorithms presented in this chapter) are based on optimization of the basic c-means objective function, or some modification of it. Hence we start our discussion with presenting the fuzzy c-means functional.

[2] The term "possibilistic" (partition, clustering, etc.) has been introduced in (Krishnapuram and Keller, 1993). In the literature, the terms "constrained fuzzy partition" and "unconstrained fuzzy partition" are also used to denote partitions (4.4) and (4.5), respectively.

4.3.1. The Fuzzy c-Means Functional

A large family of fuzzy clustering algorithms is based on minimization of the *fuzzy c-means* functional formulated as (Dunn, 1974; Bezdek, 1981):

$$J(\mathbf{Z};\mathbf{U},\mathbf{V}) = \sum_{i=1}^{c}\sum_{k=1}^{N}(\mu_{ik})^m \|\mathbf{z}_k - \mathbf{v}_i\|_{\mathbf{A}}^2 \tag{4.6a}$$

where

$$\mathbf{U} = [\mu_{ik}] \in M_{fc} \tag{4.6b}$$

is a fuzzy partition matrix of $\mathbf{Z}$,

$$\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c], \quad \mathbf{v}_i \in \mathbb{R}^n \tag{4.6c}$$

is a vector of *cluster prototypes* (centers), which have to be determined,

$$D_{ik\mathbf{A}}^2 = \|\mathbf{z}_k - \mathbf{v}_i\|_{\mathbf{A}}^2 = (\mathbf{z}_k - \mathbf{v}_i)^T\mathbf{A}(\mathbf{z}_k - \mathbf{v}_i) \tag{4.6d}$$

is a squared inner-product distance norm, and

$$m \in [1, \infty) \tag{4.6e}$$

is a parameter which determines the fuzziness of the resulting clusters. The value of the cost function (4.6a) can be seen as a measure of the total variance of $\mathbf{z}_k$ from $\mathbf{v}_i$.

4.3.2. The Fuzzy c-Means Algorithm

The minimization of the c-means functional (4.6a) represents a nonlinear optimization problem that can be solved by using a variety of methods, including iterative minimization, simulated annealing or genetic algorithms. The most popular method is a simple Picard iteration through the first-order conditions for stationary points of (4.6a), known as the fuzzy c-means (FCM) algorithm.

The stationary points of the objective function (4.6a) can be found by adjoining the constraint (4.4b) to J by means of Lagrange multipliers:

$$\bar{J}(\mathbf{Z};\mathbf{U},\mathbf{V},\boldsymbol{\lambda}) = \sum_{i=1}^{c}\sum_{k=1}^{N}(\mu_{ik})^m D_{ik\mathbf{A}}^2 + \sum_{k=1}^{N}\lambda_k\left[\sum_{i=1}^{c}\mu_{ik} - 1\right], \tag{4.7}$$

and by setting the gradients of $\bar{J}$ with respect to $\mathbf{U}$, $\mathbf{V}$ and $\boldsymbol{\lambda}$ to zero. It can be shown that if $D_{ik\mathbf{A}}^2 > 0, \forall i, k$ and $m > 1$, then $(\mathbf{U},\mathbf{V}) \in M_{fc} \times \mathbb{R}^{n\times c}$ may minimize (4.6a) only if

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{c}(D_{ik\mathbf{A}}/D_{jk\mathbf{A}})^{2/(m-1)}}, \quad 1 \le i \le c, \quad 1 \le k \le N, \tag{4.8a}$$

and

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N}(\mu_{ik})^m\mathbf{z}_k}{\sum_{k=1}^{N}(\mu_{ik})^m}; \quad 1 \le i \le c\,. \tag{4.8b}$$

This solution also satisfies the remaining constraints (4.4a) and (4.4c). Equations (4.8) are first-order necessary conditions for stationary points of the functional (4.6a). The FCM (Algorithm 4.1) iterates through (4.8a) and (4.8b). Sufficiency of (4.8) and the convergence of the FCM algorithm is proven in (Bezdek, 1980). Note that (4.8b) gives $\mathbf{v}_i$ as the weighted mean of the data items that belong to a cluster, where the weights are the membership degrees. That is why the algorithm is called "c-means".

Algorithm 4.1 Fuzzy c-means (FCM).

Given the data set $\mathbf{Z}$, choose the number of clusters $1 < c < N$, the weighting exponent $m > 1$, the termination tolerance $\epsilon > 0$ and the norm-inducing matrix $\mathbf{A}$. Initialize the partition matrix randomly, such that $\mathbf{U}^{(0)} \in M_{fc}$.

Repeat for $l = 1, 2, \ldots$

Step 1: Compute the cluster prototypes (means):

$$\mathbf{v}_i^{(l)} = \frac{\sum_{k=1}^{N} \left(\mu_{ik}^{(l-1)}\right)^m \mathbf{z}_k}{\sum_{k=1}^{N} \left(\mu_{ik}^{(l-1)}\right)^m}, \quad 1 \leq i \leq c\,.$$

Step 2: Compute the distances:

$$D_{ik\mathbf{A}}^2 = (\mathbf{z}_k - \mathbf{v}_i^{(l)})^T \mathbf{A} (\mathbf{z}_k - \mathbf{v}_i^{(l)}), \quad 1 \leq i \leq c, \quad 1 \leq k \leq N\,.$$

Step 3: Update the partition matrix:
for $1 \leq k \leq N$
if $D_{ik\mathbf{A}} > 0$ for all $i = 1, 2, \ldots, c$

$$\mu_{ik}^{(l)} = \frac{1}{\sum_{j=1}^{c} (D_{ik\mathbf{A}} / D_{jk\mathbf{A}})^{2/(m-1)}},$$

otherwise

$$\mu_{ik}^{(l)} \begin{cases} = 0 & \text{if } D_{ik\mathbf{A}} > 0 \\ \in [0,1] & \text{if } D_{ik\mathbf{A}} = 0 \end{cases} \qquad \text{with } \sum_{i=1}^{c} \mu_{ik}^{(l)} = 1\,.$$

until $\|\mathbf{U}^{(l)} - \mathbf{U}^{(l-1)}\| < \epsilon$.

Some remarks should be made:

1. The FCM algorithm converges to a *local* minimum of the c-means functional (4.6a). Hence, different initializations may lead to different results.

2. While steps 1 and 2 are straightforward, step 3 is a bit more complicated, as a singularity in FCM occurs when $D_{ik\mathbf{A}} = 0$ for some $\mathbf{z}_k$ and one or more $\mathbf{v}_i$. In this case, the membership degree in (4.8a) cannot be computed. When this happens (rare in practice), zero membership is assigned to the clusters for which $D_{ik\mathbf{A}} > 0$ and the memberships are distributed arbitrarily among the clusters for which $D_{ik\mathbf{A}} = 0$, such that the constraint in (4.4b) is satisfied.

3. The alternating optimization scheme used by FCM loops through the estimates $\mathbf{U}^{(l-1)} \to \mathbf{V}^{(l)} \to \mathbf{U}^{(l)}$ and terminates as soon as $\|\mathbf{U}^{(l)} - \mathbf{U}^{(l-1)}\| < \epsilon$. Alternatively, the algorithm can be initialized with $\mathbf{V}^{(0)}$, loop through $\mathbf{V}^{(l-1)} \to \mathbf{U}^{(l)} \to \mathbf{V}^{(l)}$, and terminate on $\|\mathbf{V}^{(l)} - \mathbf{V}^{(l-1)}\| < \epsilon$. The error norm in the termination criterion is usually chosen as $\max_{ik}(|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}|)$. Different results may be obtained with the same values of ϵ, since the termination criterion used in Algorithm 4.1 requires that more parameters become close to one another.

4.3.3. Parameters of the FCM Algorithm

Before using the FCM algorithm, the following parameters must be specified: the number of clusters, c, the 'fuzziness' exponent, m, the termination tolerance, ϵ, and the norm-inducing matrix, $\mathbf{A}$. Moreover, the fuzzy partition matrix, $\mathbf{U}$, must be initialized. The choices for these parameters are now described one by one.

Number of Clusters

The number of clusters c is the most important parameter, in the sense that the remaining parameters have less influence on the resulting partition. When clustering real data without any a priori information about the structures in the data, one usually has to make assumptions about the number of underlying clusters. The chosen clustering algorithm then searches for c clusters, regardless of whether they are really present in the data or not. Two main approaches to determining the appropriate number of clusters in data can be distinguished:

1. *Validity measures.* Validity measures are scalar indices that assess the goodness of the obtained partition. Clustering algorithms generally aim at locating well-separated and compact clusters. When the number of clusters is chosen equal to the number of groups that actually exist in the data, it can be expected that the clustering algorithm will identify them correctly. When this is not the case, misclassifications appear, and the clusters are not likely to be well separated and compact. Hence, most cluster validity measures are designed to quantify the separation and the compactness of the clusters. However, as Bezdek (1981) points out, the concept of cluster validity is open to interpretation and can be formulated in different ways. Consequently, many validity measures have been introduced in the literature, see (Bezdek, 1981; Gath and Geva, 1989; Pal and Bezdek, 1995) among others. For the FCM algorithm, the

Xie-Beni index (Xie and Beni, 1991)

$$\chi(\mathbf{Z};\mathbf{U},\mathbf{V}) = \frac{\sum_{i=1}^{c}\sum_{k=1}^{N}\mu_{ik}^{m} \parallel \mathbf{z}_k - \mathbf{v}_i \parallel^2}{c \cdot \min_{i\neq j}\Big(\parallel \mathbf{v}_i - \mathbf{v}_j \parallel^2\Big)} \tag{4.9}$$

has been found to perform well in practice. This index can be interpreted as the ratio of the total within-group variance and the separation of the cluster centers. The best partition minimizes the value of $\chi(\mathbf{Z};\mathbf{U},\mathbf{V})$.

2. *Iterative merging or insertion of clusters.* The basic idea of cluster merging is to start with a sufficiently large number of clusters, and successively reduce this number by merging clusters that are similar (compatible) with respect to some well-defined criteria (Krishnapuram and Freg, 1992; Kaymak and Babuška, 1995). One can also adopt an opposite approach, i.e., start with a small number of clusters and iteratively insert clusters in the regions where the data points have low degree of membership in the existing clusters (Gath and Geva, 1989).

Fuzziness Parameter

The weighting exponent m is a rather important parameter as well, because it significantly influences the fuzziness of the resulting partition. As m approaches one from above, the partition becomes hard ($\mu_{ik} \in \{0,1\}$) and $\mathbf{v}_i$ are ordinary means of the clusters. As $m \to \infty$, the partition becomes completely fuzzy ($\mu_{ik} = 1/c$) and the cluster means are all equal to the mean of $\mathbf{Z}$. These limit properties of (4.6) are independent of the optimization method used (Pal and Bezdek, 1995). Usually, $m = 2$ is initially chosen.

Termination Criterion

The FCM algorithm stops iterating when the norm of the difference between $\mathbf{U}$ in two successive iterations is smaller than the termination parameter ϵ. For the maximum norm $\max_{ik}(|\mu_{ik}^{(l)} - \mu_{ik}^{(l-1)}|)$, the usual choice is $\epsilon = 0.001$, even though $\epsilon = 0.01$ works well in most cases, while drastically reducing the computing times.

Norm-Inducing Matrix

The shape of the clusters is determined by the choice of the matrix $\mathbf{A}$ in the distance measure (4.6d). A common choice is $\mathbf{A} = \mathbf{I}$, which gives the standard Euclidean norm:

$$D_{ik}^2 = (\mathbf{z}_k - \mathbf{v}_i)^T(\mathbf{z}_k - \mathbf{v}_i). \tag{4.10}$$

Another choice for $\mathbf{A}$ is a diagonal matrix that accounts for different variances in the directions of the coordinate axes of $\mathbf{Z}$:

$$\mathbf{A} = \begin{bmatrix} (1/\sigma_1)^2 & 0 & \cdots & 0 \\ 0 & (1/\sigma_2)^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (1/\sigma_n)^2 \end{bmatrix}. \tag{4.11}$$

This matrix induces a diagonal norm on $\mathbb{R}^n$. Finally, $\mathbf{A}$ can be defined as the inverse of the covariance matrix of $\mathbf{Z}$: $\mathbf{A} = \mathbf{R}^{-1}$, with

$$\mathbf{R} = \frac{1}{N} \sum_{k=1}^{N} (\mathbf{z}_k - \bar{\mathbf{z}})(\mathbf{z}_k - \bar{\mathbf{z}})^T . \tag{4.12}$$

Here $\bar{\mathbf{z}}$ denotes the mean of the data. In this case, $\mathbf{A}$ induces the Mahalanobis norm on $\mathbb{R}^n$.

The norm influences the clustering criterion by changing the measure of dissimilarity. The Euclidean norm induces hyperspherical clusters (surfaces of constant membership are hyperspheres). Both the diagonal and the Mahalanobis norm generate hyperellipsoidal clusters. With the diagonal norm, the axes of the hyperellipsoids are parallel to the coordinate axes, while with the Mahalanobis norm the orientation of the hyperellipsoid is arbitrary, as shown in Figure 4.3.

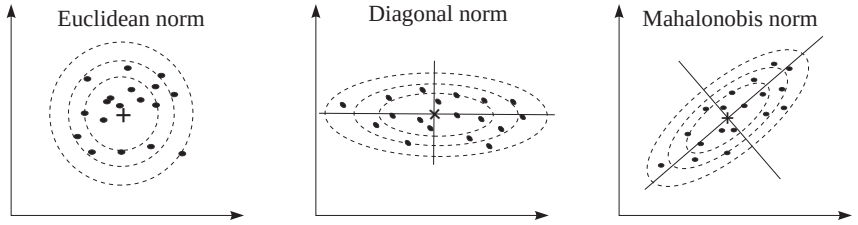


Figure 4.3.: Different distance norms used in fuzzy clustering.

A common limitation of clustering algorithms based on a fixed distance norm is that such a norm forces the objective function to prefer clusters of a certain shape even if they are not present in the data. This is demonstrated by the following example.

Example 4.4 ***Fuzzy* c*-means clustering.*** Consider a synthetic data set in $\mathbb{R}^2$, which contains two well-separated clusters of different shapes, as depicted in Figure 4.4. The samples in both clusters are drawn from the normal distribution. The standard deviation for the upper cluster is 0.2 for both axes, whereas in the lower cluster it is 0.2 for the horizontal axis and 0.05 for the vertical axis. The FCM algorithm was applied to this data set. The norm-inducing matrix was set to $\mathbf{A} = \mathbf{I}$ for both clusters, the weighting exponent to $m = 2$, and the termination criterion to $\epsilon = 0.01$. The algorithm was initialized with a random partition matrix and converged after 4 iterations. From the membership level curves in Figure 4.4, one can see that the FCM algorithm imposes a circular shape on both clusters, even though the lower cluster is rather elongated.

Note that it is of no help to use another $\mathbf{A}$, since the two clusters have different shapes. Generally, different matrices $\mathbf{A}_i$ are required, but there is no guideline as to how to choose them a priori. In Section 4.4, we will see that these matrices can be adapted by using estimates of the data covariance. A partition obtained with the Gustafson–Kessel algorithm, which uses such an adaptive distance norm, is presented in Example 4.5.

□

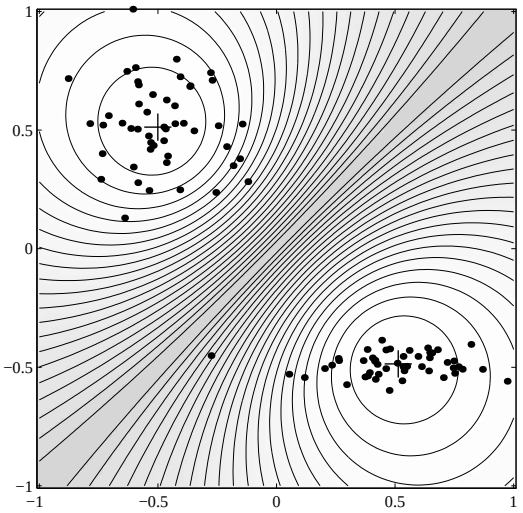


Figure 4.4.: The fuzzy c-means algorithm imposes a spherical shape on the clusters, regardless of the actual data distribution. The dots represent the data points, '+' are the cluster means. Also shown are level curves of the clusters. Dark shading corresponds to membership degrees around 0.5.

Initial Partition Matrix

The partition matrix is usually initialized at random, such that $\mathbf{U} \in M_{fc}$. A simple approach to obtain such $\mathbf{U}$ is to initialize the cluster centers $\mathbf{v}_i$ at random and compute the corresponding $\mathbf{U}$ by (4.8a) (i.e., by using the third step of the FCM algorithm).

4.3.4. Extensions of the Fuzzy c-Means Algorithm

There are several well-known extensions of the basic c-means algorithm:

- Algorithms using an adaptive distance measure, such as the Gustafson–Kessel algorithm (Gustafson and Kessel, 1979) and the fuzzy maximum likelihood estimation algorithm (Gath and Geva, 1989).
- Algorithms based on hyperplanar or functional prototypes, or prototypes defined by functions. They include the fuzzy c-varieties (Bezdek, 1981), fuzzy c-elliptotypes (Bezdek et al., 1981), and fuzzy regression models (Hathaway and Bezdek, 1993).
- Algorithms that search for possibilistic partitions in the data, i.e., partitions where the constraint (4.4b) is relaxed.

In the following sections we will focus on the Gustafson–Kessel algorithm.

4.4. Gustafson–Kessel Algorithm

Gustafson and Kessel (1979) extended the standard fuzzy c-means algorithm by employing an adaptive distance norm, in order to detect clusters of different geometrical shapes in

one data set. Each cluster has its own norm-inducing matrix $\mathbf{A}_i$, which yields the following inner-product norm:

$$D^2_{ik\mathbf{A}_i} = (\mathbf{z}_k - \mathbf{v}_i)^T \mathbf{A}_i (\mathbf{z}_k - \mathbf{v}_i)\,. \tag{4.13}$$

The matrices $\mathbf{A}_i$ are used as optimization variables in the c-means functional, thus allowing each cluster to adapt the distance norm to the local topological structure of the data. The objective functional of the GK algorithm is defined by:

$$J(\mathbf{Z}; \mathbf{U}, \mathbf{V}, \{\mathbf{A}_i\}) = \sum_{i=1}^{c} \sum_{k=1}^{N} (\mu_{ik})^m D^2_{ik\mathbf{A}_i} \tag{4.14}$$

This objective function cannot be directly minimized with respect to $\mathbf{A}_i$, since it is linear in $\mathbf{A}_i$. To obtain a feasible solution, $\mathbf{A}_i$ must be constrained in some way. The usual way of accomplishing this is to constrain the determinant of $\mathbf{A}_i$:

$$|\mathbf{A}_i| = \rho_i, \quad \rho_i > 0, \quad \forall i\,. \tag{4.15}$$

Allowing the matrix $\mathbf{A}_i$ to vary with its determinant fixed corresponds to optimizing the cluster's shape while its volume remains constant. By using the Lagrange-multiplier method, the following expression for $\mathbf{A}_i$ is obtained (Gustafson and Kessel, 1979):

$$\mathbf{A}_i = [\rho_i \det(\mathbf{F}_i)]^{1/n} \mathbf{F}_i^{-1}, \tag{4.16}$$

where $\mathbf{F}_i$ is the *fuzzy covariance matrix* of the ith cluster given by

$$\mathbf{F}_i = \frac{\sum_{k=1}^{N} (\mu_{ik})^m (\mathbf{z}_k - \mathbf{v}_i)(\mathbf{z}_k - \mathbf{v}_i)^T}{\sum_{k=1}^{N} (\mu_{ik})^m}\,. \tag{4.17}$$

Note that the substitution of (4.16) and (4.17) into (4.13) gives a generalized squared Mahalanobis distance norm, where the covariance is weighted by the membership degrees in $\mathbf{U}$. The GK algorithm is given in Algorithm 4.2 and its MATLAB implementation can be found in the Appendix. The GK algorithm is computationally more involved than FCM, since the inverse and the determinant of the cluster covariance matrix must be calculated in each iteration.

Algorithm 4.2 Gustafson–Kessel (GK) algorithm.

Given the data set $\mathbf{Z}$, choose the number of clusters $1 < c < N$, the weighting exponent $m > 1$ and the termination tolerance $\epsilon > 0$ and the cluster volumes ρ_i. Initialize the partition matrix randomly, such that $\mathbf{U}^{(0)} \in M_{fc}$.

Repeat for $l = 1, 2, \ldots$

Step 1: Compute cluster prototypes (means):

$$\mathbf{v}_i^{(l)} = \frac{\sum_{k=1}^{N} \left(\mu_{ik}^{(l-1)}\right)^m \mathbf{z}_k}{\sum_{k=1}^{N} \left(\mu_{ik}^{(l-1)}\right)^m}, \quad 1 \leq i \leq c\,.$$

Step 2: Compute the cluster covariance matrices:

$$\mathbf{F}_i = \frac{\sum_{k=1}^{N} \left(\mu_{ik}^{(l-1)}\right)^m \left(\mathbf{z}_k - \mathbf{v}_i^{(l)}\right)\left(\mathbf{z}_k - \mathbf{v}_i^{(l)}\right)^T}{\sum_{k=1}^{N} \left(\mu_{ik}^{(l-1)}\right)^m}, \quad 1 \leq i \leq c\,.$$

Step 3: Compute the distances:

$$D_{ik\mathbf{A}_i}^2 = \left(\mathbf{z}_k - \mathbf{v}_i^{(l)}\right)^T \left[\left[\rho_i \det(\mathbf{F}_i)\right]^{1/n} \mathbf{F}_i^{-1}\right] \left(\mathbf{z}_k - \mathbf{v}_i^{(l)}\right), \quad 1 \leq i \leq c, \quad 1 \leq k \leq N\,.$$

Step 4: Update the partition matrix:
for $1 \leq k \leq N$
if $D_{ik\mathbf{A}_i} > 0$ for all $i = 1, 2, \ldots, c$

$$\mu_{ik}^{(l)} = \frac{1}{\sum_{j=1}^{c} (D_{ik\mathbf{A}_i} / D_{jk\mathbf{A}_i})^{2/(m-1)}},$$

otherwise

$$\mu_{ik}^{(l)} \begin{cases} = 0 & \text{if } D_{ik\mathbf{A}_i} > 0 \\ \in [0,1] & \text{if } D_{ik\mathbf{A}_i} = 0 \end{cases} \qquad \text{with} \quad \sum_{i=1}^{c} \mu_{ik}^{(l)} = 1\,.$$

until $\|\mathbf{U}^{(l)} - \mathbf{U}^{(l-1)}\| < \epsilon$.

4.4.1. Parameters of the Gustafson–Kessel Algorithm

The same parameters must be specified as for the FCM algorithm (except for the norm-inducing matrix $\mathbf{A}$, which is adapted automatically): the number of clusters c, the 'fuzziness' exponent m, the termination tolerance ϵ. Additional parameters are the cluster volumes ρ_i.

Without any prior knowledge, ρ_i is simply fixed at 1 for each cluster. A drawback of the this setting is that due to the constraint (4.15), the GK algorithm only can find clusters of approximately equal volumes.

4.4.2. Interpretation of the Cluster Covariance Matrices

The eigenstructure of the cluster covariance matrix $\mathbf{F}_i$ provides information about the shape and orientation of the cluster. The ratio of the lengths of the cluster's hyperellipsoid axes is given by the ratio of the square roots of the eigenvalues of $\mathbf{F}_i$. The directions of the axes are given by the eigenvectors of $\mathbf{F}_i$, as shown in Figure 4.5. The GK algorithm can be used to detect clusters along linear subspaces of the data space. These clusters are represented by flat hyperellipsoids, which can be regarded as hyperplanes. The eigenvector corresponding to the smallest eigenvalue determines the normal to the hyperplane, and can be used to compute optimal local linear models from the covariance matrix.

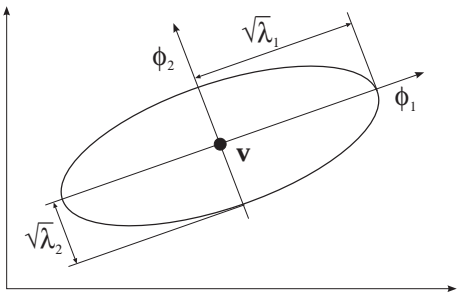


Figure 4.5.: Equation $(\mathbf{z}-\mathbf{v})^T\mathbf{F}^{-1}(\mathbf{x}-\mathbf{v}) = 1$ defines a hyperellipsoid. The length of the jth axis of this hyperellipsoid is given by $\sqrt{\lambda}_j$ and its direction is spanned by $\boldsymbol{\phi}_j$, where λ_j and $\boldsymbol{\phi}_j$ are the jth eigenvalue and the corresponding eigenvector of $\mathbf{F}$, respectively.

Example 4.5 ***Gustafson–Kessel algorithm.*** The GK algorithm was applied to the data set from Example 4.4, using the same initial settings as the FCM algorithm. Figure 4.4 shows that the GK algorithm can adapt the distance norm to the underlying distribution of the data. One nearly circular cluster and one elongated ellipsoidal cluster are obtained. The shape of the clusters can be determined from the eigenstructure of the resulting covariance matrices $\mathbf{F}_i$. The eigenvalues of the clusters are given in Table 4.1.

Table 4.1.: Eigenvalues of the cluster covariance matrices for clusters in Figure 4.6.

cluster	λ_1	λ_2	$\sqrt{\lambda}_1/\sqrt{\lambda}_2$
upper	0.0352	0.0310	1.0666
lower	0.0482	0.0028	4.1490

One can see that the ratios given in the last column reflect quite accurately the ratio of the standard deviations in each data group (1 and 4 respectively). For the lower cluster, the

unitary eigenvector corresponding to λ_2, $\boldsymbol{\phi}_2 = [0.0134, 0.9999]^T$, can be seen as a normal to a line representing the second cluster's direction, and it is, indeed, nearly parallel to the vertical axis.

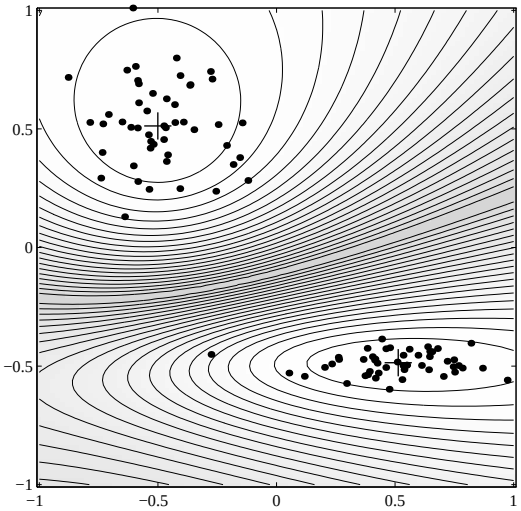


Figure 4.6.: The Gustafson–Kessel algorithm can detect clusters of different shape and orientation. The points represent the data, '+' are the cluster means. Also shown are level curves of the clusters. Dark shading corresponds to membership degrees around 0.5.

□

4.5. Summary and Concluding Remarks

Fuzzy clustering is a powerful unsupervised method for the analysis of data and construction of models. In this chapter, an overview of the most frequently used fuzzy clustering algorithms has been given. It has been shown that the basic c-means iterative scheme can be used in combination with adaptive distance measures to reveal clusters of various shapes. The choice of the important user-defined parameters, such as the number of clusters and the fuzziness parameter, has been discussed.

4.6. Problems

1. State the definitions and discuss the differences of fuzzy and non-fuzzy (hard) partitions. Give an example of a fuzzy and non-fuzzy partition matrix. What are the advantages of fuzzy clustering over hard clustering?

2. State mathematically at least two different distance norms used in fuzzy clustering. Explain the differences between them.

3. Name two fuzzy clustering algorithms and explain how they differ from each other.

4. State the fuzzy c-mean functional and explain all symbols.

5. Outline the steps required in the initialization and execution of the fuzzy c-means algorithm. What is the role and the effect of the user-defined parameters in this algorithm?

5. Construction Techniques for Fuzzy Systems

Two common sources of information for building fuzzy systems are *prior knowledge* and *data* (measurements). Prior knowledge tends to be of a rather approximate nature (qualitative knowledge, heuristics), which usually originates from "experts", i.e., process designers, operators, etc. In this sense, fuzzy models can be regarded as simple *fuzzy expert systems* (Zimmermann, 1987).

For many processes, data are available as records of the process operation or special identification experiments can be designed to obtain the relevant data. Building fuzzy models from data involves methods based on fuzzy logic and approximate reasoning, but also ideas originating from the field of neural networks, data analysis and conventional systems identification. The acquisition or tuning of fuzzy models by means of data is usually termed *fuzzy systems identification*.

Two main approaches to the integration of knowledge and data in a fuzzy model can be distinguished:

1. The expert knowledge expressed in a verbal form is translated into a collection of if–then rules. In this way, a certain model structure is created. Parameters in this structure (membership functions, consequent singletons or parameters of the TS consequents) can be fine-tuned using input-output data. The particular tuning algorithms exploit the fact that at the computational level, a fuzzy model can be seen as a layered structure (network), similar to artificial neural networks, to which standard learning algorithms can be applied. This approach is usually termed *neuro-fuzzy modeling*.

2. No prior knowledge about the system under study is initially used to formulate the rules, and a fuzzy model is constructed from data. It is expected that the extracted rules and membership functions can provide an a posteriori interpretation of the system's behavior. An expert can confront this information with his own knowledge, can modify the rules, or supply new ones, and can design additional experiments in order to obtain more informative data. This approach can be termed *rule extraction*. Fuzzy clustering is one of the techniques that are often applied.(Yoshinari et al., 1993; Nakamori and Ryoke, 1994; Babuška and Verbruggen, 1997)

These techniques, of course, can be combined, depending on the particular application. In the sequel, we describe the main steps and choices in the knowledge-based construction of fuzzy models, and the main techniques to extract or fine-tune fuzzy models by means of data.

5.1. Structure and Parameters

With regard to the design of fuzzy (and also other) models, two basic items are distinguished: the structure and the parameters of the model. The structure determines the flexibility of the model in the approximation of (unknown) mappings. The parameters are then tuned (estimated) to fit the data at hand. A model with a rich structure is able to approximate more complicated functions, but, at the same time, has worse *generalization* properties. Good generalization means that a model fitted to one data set will also perform well on another data set from the same process. In fuzzy models, structure selection involves the following choices:

- *Input and output variables.* With complex systems, it is not always clear which variables should be used as inputs to the model. In the case of dynamic systems, one also must estimate the order of the system. For the input-output NARX (nonlinear autoregressive with exogenous input) model (3.58) this means to define the number of input and output lags n_y and n_u, respectively. Prior knowledge, insight in the process behavior and the purpose of modeling are the typical sources of information for this choice. Sometimes, automatic data-driven selection can be used to compare different choices in terms of some performance criteria.

- *Structure of the rules.* This choice involves the model type (linguistic, singleton, relational, Takagi-Sugeno) and the antecedent form (refer to Section 3.2.6). Important aspects are the purpose of modeling and the type of available knowledge.

- *Number and type of membership functions for each variable.* This choice determines the level of detail (granularity) of the model. Again, the purpose of modeling and the detail of available knowledge, will influence this choice. Automated, data-driven methods can be used to add or remove membership functions from the model.

- *Type of the inference mechanism, connective operators, defuzzification method.* These choices are restricted by the type of fuzzy model (Mamdani, TS). Within these restrictions, however, some freedom remains, e.g., as to the choice of the conjunction operators, etc. To facilitate data-driven optimization of fuzzy models (learning), differentiable operators (product, sum) are often preferred to the standard min and max operators.

After the structure is fixed, the performance of a fuzzy model can be fine-tuned by adjusting its parameters. Tunable parameters of linguistic models are the parameters of antecedent and consequent membership functions (determine their shape and position) and the rules (determine the mapping between the antecedent and consequent fuzzy regions). In fuzzy relational models, this mapping is encoded in the fuzzy relation. Takagi-Sugeno models have parameters in antecedent membership functions and in the consequent functions ($\mathbf{a}$ and b for the affine TS model).

5.2. Knowledge-Based Design

To design a (linguistic) fuzzy model based on available expert knowledge, the following steps can be followed:

1. Select the input and output variables, the structure of the rules, and the inference and defuzzification methods.
2. Decide on the number of linguistic terms for each variable and define the corresponding membership functions.
3. Formulate the available knowledge in terms of fuzzy if-then rules.
4. Validate the model (typically by using data). If the model does not meet the expected performance, iterate on the above design steps.

This procedure is very similar to the heuristic design of fuzzy controllers (Section 6.3.4). It should be noted that the success of the knowledge-based design heavily depends on the problem at hand, and the extent and quality of the available knowledge. For some problems, it may lead fast to useful models, while for others it may be a very time-consuming and inefficient procedure (especially manual fine-tuning of the model parameters). Therefore, it is useful to combine the knowledge based design with a data-driven tuning of the model parameters. The following sections review several methods for the adjustment of fuzzy model parameters by means of data.

5.3. Data-Driven Acquisition and Tuning of Fuzzy Models

The strong potential of fuzzy models lies in their ability to combine heuristic knowledge expressed in the form of rules with information obtained from measured data. Various estimation and optimization techniques for the parameters of fuzzy models are presented in the sequel.

Assume that a set of N input-output data pairs $\{(\mathbf{x}_i, y_i) \mid i = 1, 2, \ldots, N\}$ is available for the construction of a fuzzy system. Recall that $\mathbf{x}_i \in \mathbb{R}^p$ are input vectors and y_i are output scalars. Denote $\mathbf{X} \in \mathbb{R}^{N \times p}$ a matrix having the vectors $\mathbf{x}_k^T$ in its rows, and $\mathbf{y} \in \mathbb{R}^N$ a vector containing the outputs y_k:

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T, \quad \mathbf{y} = [y_1, \ldots, y_N]^T. \tag{5.1}$$

In the following sections, the estimation of consequent and antecedent parameters is addressed.

5.3.1. Least-Squares Estimation of Consequents

The defuzzification formulas of the singleton and TS models are linear in the consequent parameters, $\mathbf{a}_i$, b_i (see (3.38) and (3.53), respectively). Hence, these parameters can be estimated from the available data by least-squares techniques. Denote $\mathbf{\Gamma}_i \in \mathbb{R}^{N \times N}$ the diagonal matrix having the normalized membership degree $\gamma_i(\mathbf{x}_k)$ as its kth diagonal element. By appending a unitary column to $\mathbf{X}$, the extended matrix $\mathbf{X}_e = [\mathbf{X}, \mathbf{1}]$ is created. Further, denote $\mathbf{X}'$ the matrix in $\mathbb{R}^{N \times K(p+1)}$ composed of the products of matrices $\mathbf{\Gamma}_i$ and $\mathbf{X}_e$

$$\mathbf{X}' = [\mathbf{\Gamma}_1 \mathbf{X}_e,\ \mathbf{\Gamma}_2 \mathbf{X}_e,\ \ldots,\ \mathbf{\Gamma}_K \mathbf{X}_e]\ . \tag{5.2}$$

The consequent parameters $\mathbf{a}_i$ and b_i are lumped into a single parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{K(p+1)}$:

$$\boldsymbol{\theta} = \left[\mathbf{a}_1^T, b_1, \mathbf{a}_2^T, b_2, \ldots, \mathbf{a}_K^T, b_K\right]^T . \tag{5.3}$$

Given the data $\mathbf{X}$, $\mathbf{y}$, eq. (3.53) now can be written in a matrix form, $\mathbf{y} = \mathbf{X}'\boldsymbol{\theta} + \epsilon$. It is well known that this set of equations can be solved for the parameter $\boldsymbol{\theta}$ by:

$$\boldsymbol{\theta} = \left[(\mathbf{X}')^T\mathbf{X}'\right]^{-1}(\mathbf{X}')^T\mathbf{y} . \tag{5.4}$$

This is an optimal least-squares solution which gives the minimal prediction error, and as such is suitable for prediction models. At the same time, however, it may bias the estimates of the consequent parameters as parameters of local models. If an accurate estimate of local model parameters is desired, a weighted least-squares approach applied per rule may be used:

$$[\mathbf{a}_i^T, b_i]^T = \left[\mathbf{X}_e^T\boldsymbol{\Gamma}_i\mathbf{X}_e\right]^{-1}\mathbf{X}_e^T\boldsymbol{\Gamma}_i\mathbf{y} . \tag{5.5}$$

In this case, the consequent parameters of individual rules are estimated independently of each other, and therefore are not "biased" by the interactions of the rules. By omitting $\mathbf{a}_i$ for all $1 \leq i \leq K$, and by setting $\mathbf{X}_e = \mathbf{1}$, (5.4) and (5.5) directly apply to the singleton model (3.37).

5.3.2. Template-Based Modeling

With this approach, the domains of the antecedent variables are simply partitioned into a specified number of equally spaced and shaped membership functions. The rule base is then established to cover all the combinations of the antecedent terms. The consequent parameters are estimated by the least-squares method.

Example 5.1 Consider a nonlinear dynamic system described by a first-order difference equation:

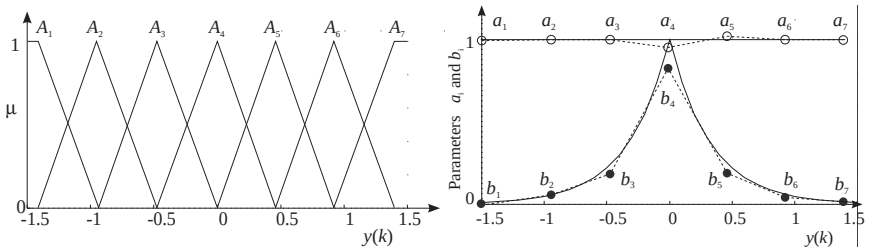
$$y(k+1) = y(k) + u(k)e^{-3|y(k)|} . \tag{5.6}$$

We use a stepwise input signal to generate with this equation a set of 300 input–output data pairs (see Figure 5.2(a)). Suppose that it is known that the system is of first order and that the nonlinearity of the system is only caused by y, the following TS rule structure can be chosen:

$$\textbf{If } y(k) \text{ is } A_i \textbf{ then } y(k+1) = a_i y(k) + b_i u(k) . \tag{5.7}$$

Assuming that no further prior knowledge is available, seven equally spaced triangular membership functions, A_1 to A_7, are defined in the domain of $y(k)$, as shown in Figure 5.1(a).

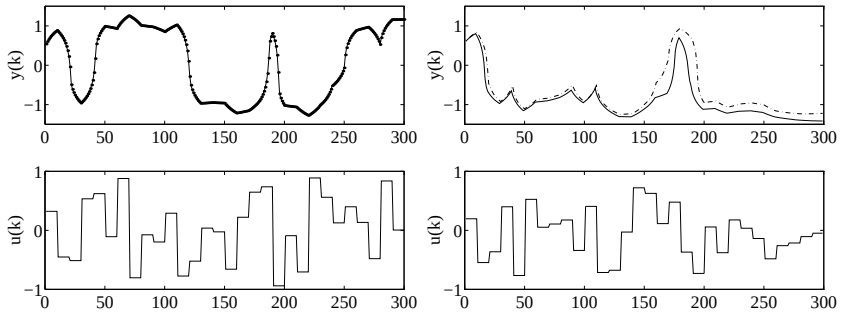
The consequent parameters can be estimated by the least-squares method. Figure 5.1(b) shows a plot of the parameters a_i, b_i against the cores of the antecedent fuzzy sets A_i. Also plotted is the linear interpolation between the parameters (dashed line) and the true system nonlinearity (solid line). The interpolation between a_i and b_i is linear, since the membership functions are piece-wise linear (triangular). One can observe that the dependence of the consequent parameters on the antecedent variable approximates quite accurately the system's nonlinearity, which gives the model a certain transparency. Their values,



(a) Membership functions. (b) Estimated consequent parameters.

Figure 5.1.: (a) Equidistant triangular membership functions designed for the output $y(k)$; (b) comparison of the true system nonlinearity (solid line) and its approximation in terms of the estimated consequent parameters (dashed line).

$\mathbf{a}^T = [1.00, 1.00, 1.00, 0.97, 1.01, 1.00, 1.00]$ and $\mathbf{b}^T = [0.01, 0.05, 0.20, 0.81, 0.20, 0.05, 0.01]^T$, indicate the strong input nonlinearity and the linear dynamics of (5.6). Validation of the model in simulation using a different data set is given in Figure 5.2(b).



(a) Identification data. (b) Validation.

Figure 5.2.: Identification data set (a), and performance of the model on a validation data set (b). Solid line: process, dashed-dotted line: model.

□

The transparent local structure of the TS model facilitates the combination of local models obtained by parameter estimation and linearization of known mechanistic (white-box) models. If measurements are available only in certain regions of the process' operating domain, parameters for the remaining regions can be obtained by linearizing a (locally valid) mechanistic model of the process. Suppose that this model is given by $y = f(\mathbf{x})$. Linearization around the center $\mathbf{c}_i$ of the ith rule's antecedent membership function yields the following parameters of the affine TS model (3.52):

$$\mathbf{a}_i = \left.\frac{df}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{c}_i}, \quad b_i = f(\mathbf{c}_i)\,. \tag{5.8}$$

A drawback of the template-based approach is that the number of rules in the model may grow very fast. If no knowledge is available as to which variables cause the nonlinearity of the system, all the antecedent variables are usually partitioned uniformly. However, the

complexity of the system's behavior is typically not uniform. Some operating regions can be well approximated by a single model, while other regions require rather fine partitioning. In order to obtain an efficient representation with as few rules as possible, the membership functions must be placed such that they capture the non-uniform behavior of the system. This often requires that system measurements are also used to form the membership functions, as discussed in the following sections.

5.3.3. Neuro-Fuzzy Modeling

We have seen that parameters that are linearly related to the output can be (optimally) estimated by least-squares methods. In order to optimize also the parameters which are related to the output in a nonlinear way, training algorithms known from the area of neural networks and nonlinear optimization can be employed. These techniques exploit the fact that, at the computational level, a fuzzy model can be seen as a layered structure (network), similar to artificial neural networks. Hence, this approach is usually referred to as neuro-fuzzy modeling (Jang and Sun, 1993; Brown and Harris, 1994; Jang, 1993).

Figure 5.3 gives an example of a singleton fuzzy model with two rules represented as a network. The rules are:

$$\textbf{If } x_1 \text{ is } A_{11} \textbf{ and } x_2 \text{ is } A_{21} \textbf{ then } y = b_1.$$
$$\textbf{If } x_1 \text{ is } A_{12} \textbf{ and } x_2 \text{ is } A_{22} \textbf{ then } y = b_2.$$

The nodes in the first layer compute the membership degree of the inputs in the antecedent fuzzy sets. The product nodes Π in the second layer represent the antecedent conjunction operator. The normalization node N and the summation node Σ realize the fuzzy-mean operator (3.38).

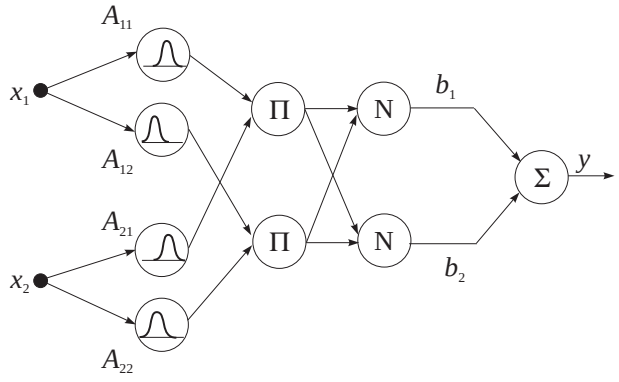


Figure 5.3.: An example of a singleton fuzzy model with two rules represented as a (neuro-fuzzy) network.

By using smooth (e.g., Gaussian) antecedent membership functions

$$\mu_{A_{ij}}(x_j; c_{ij}, \sigma_{ij}) = \exp\left(-\left(\frac{x_j - c_{ij}}{2\sigma_{ij}}\right)^2\right), \tag{5.9}$$

the c_{ij} and σ_{ij} parameters can be adjusted by gradient-descent learning algorithms, such as back-propagation (see Section 7.6.3).

5.3.4. Construction Through Fuzzy Clustering

Identification methods based on fuzzy clustering originate from data analysis and pattern recognition, where the concept of graded membership is used to represent the degree to which a given object, represented as a vector of features, is similar to some prototypical object. The degree of similarity can be calculated using a suitable distance measure. Based on the similarity, feature vectors can be clustered such that the vectors within a cluster are as similar (close) as possible, and vectors from different clusters are as dissimilar as possible (see Chapter 4).

Figure 5.4 gives an example of a data set in $\mathbb{R}^2$ clustered into two groups with prototypes $\mathbf{v}_1$ and $\mathbf{v}_2$, using the Euclidean distance measure. Fuzzy if-then rules can be extracted by projecting the clusters onto the axes.

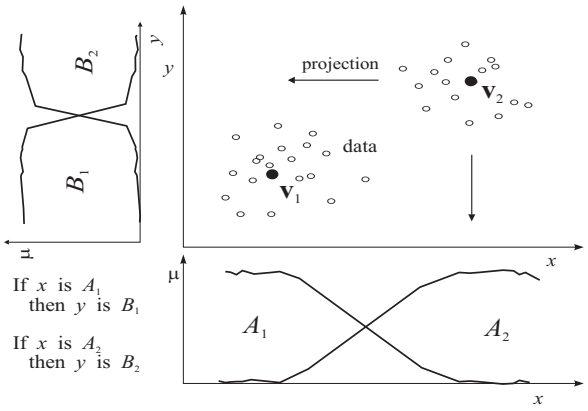


Figure 5.4.: Rule-based interpretation of fuzzy clusters.

The prototypes can also be defined as linear subspaces, (Bezdek, 1981) or the clusters can be ellipsoids with adaptively determined elliptical shape (Gustafson–Kessel algorithm, see Section 4.4). From such clusters, the antecedent membership functions and the consequent parameters of the Takagi–Sugeno model can be extracted (Figure 5.5):

$$\textbf{If } x \text{ is } A_1 \textbf{ then } y = a_1 x + b_1,$$
$$\textbf{If } x \text{ is } A_2 \textbf{ then } y = a_2 x + b_2\,.$$

Each obtained cluster is represented by one rule in the Takagi–Sugeno model. The membership functions for fuzzy sets A_1 and A_2 are generated by point-wise projection of the partition matrix onto the antecedent variables. These point-wise defined fuzzy sets are then approximated by a suitable parametric function. The consequent parameters for each rule are obtained as least-squares estimates (5.4) or (5.5).

Example 5.2 Consider a nonlinear function $y = f(x)$ defined piece-wise by:

$$\begin{aligned} y &= 0.25x, && \text{for } x \leq 3 \\ y &= (x-3)^2 + 0.75, && \text{for } 3 < x \leq 6 \\ y &= 0.25x + 8.25, && \text{for } x > 6 \end{aligned} \tag{5.10}$$

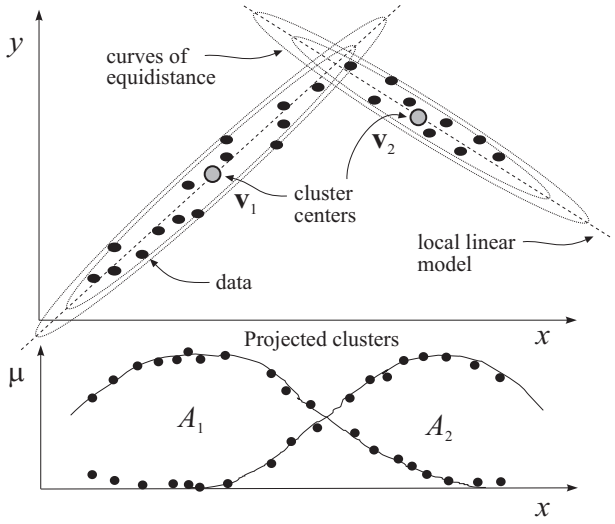
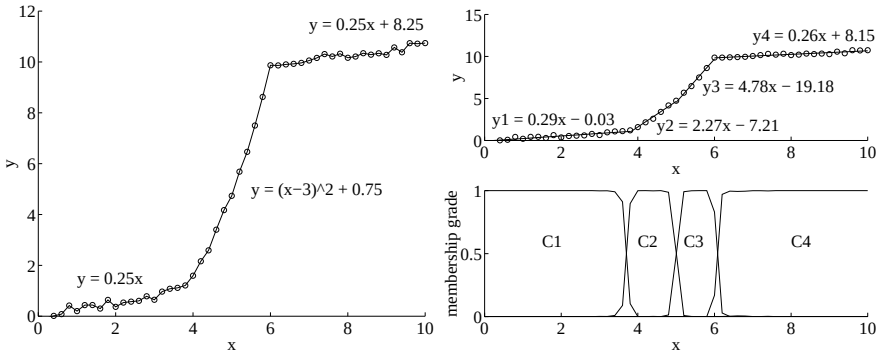


Figure 5.5.: Hyperellipsoidal fuzzy clusters.

Figure 5.6(a) shows a plot of this function evaluated in 50 samples uniformly distributed over $x \in [0, 10]$. Zero-mean, uniformly distributed noise with amplitude 0.1 was added to y.



(a) A nonlinear function (5.10). (b) Cluster prototypes and the corresponding fuzzy sets.

Figure 5.6.: Approximation of a static nonlinear function using a Takagi–Sugeno (TS) fuzzy model.

The data set $\{(x_i, y_i) \mid i = 1, 2, \ldots, 50\}$ was clustered into four hyperellipsoidal clusters. The upper plot of Figure 5.6(b) shows the local linear models obtained through clustering, the bottom plot shows the corresponding fuzzy partition. In terms of the TS rules, the fuzzy model is expressed as:

$$
\begin{aligned}
X_1\colon&\ \textbf{If } x \text{ is } C_1 \textbf{ then } y = 0.29x - 0.03\\
X_2\colon&\ \textbf{If } x \text{ is } C_2 \textbf{ then } y = 2.27x - 7.21\\
X_3\colon&\ \textbf{If } x \text{ is } C_3 \textbf{ then } y = 4.78x - 19.18\\
X_4\colon&\ \textbf{If } x \text{ is } C_4 \textbf{ then } y = 0.26x + 8.15
\end{aligned}
$$

Note that the consequents of X_1 and X_4 almost exactly correspond to the first and third

equation (5.10). Consequents of X_2 and X_3 are approximate tangents to the parabola defined by the second equation of (5.10) in the respective cluster centers.

□

The principle of identification in the product space extends to input–output dynamic systems in a straightforward way. In this case, the product space is formed by the regressors (lagged input and output data) and the regressand (the output to be predicted). As an example, assume a second-order NARX model $y(k+1) = f\left(y(k), y(k-1), u(k), u(k-1)\right)$. With the set of available measurements, $S = \{(u(j), y(j)) \mid j = 1, 2, \ldots, N_d\}$, the regressor matrix and the regressand vector are:

$$\mathbf{X} = \begin{bmatrix} y(2) & y(1) & u(2) & u(1) \\ y(3) & y(2) & u(3) & u(2) \\ \vdots & \vdots & \vdots & \vdots \\ y(N_d{-}1) & y(N_d{-}2) & u(N_d{-}1) & y(N_d{-}2) \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} y(3) \\ y(4) \\ \vdots \\ y(N_d) \end{bmatrix}.$$

In this example, $N = N_d - 2$. The unknown nonlinear function $y = f(\mathbf{x})$ represents a nonlinear (hyper)surface in the product space: $(X \times Y) \subset \mathbb{R}^{p+1}$. This surface is called the *regression surface*. The available data represents a sample from the regression surface. By clustering the data, local linear models can be found that approximate the regression surface.

Example 5.3 For low-order systems, the regression surface can be visualized. As an example, consider a series connection of a static dead-zone/saturation nonlinearity with a first-order linear dynamic system:

$$y(k+1) = 0.6y(k) + w(k), \tag{5.11a}$$

where $w = f(u)$ is given by:

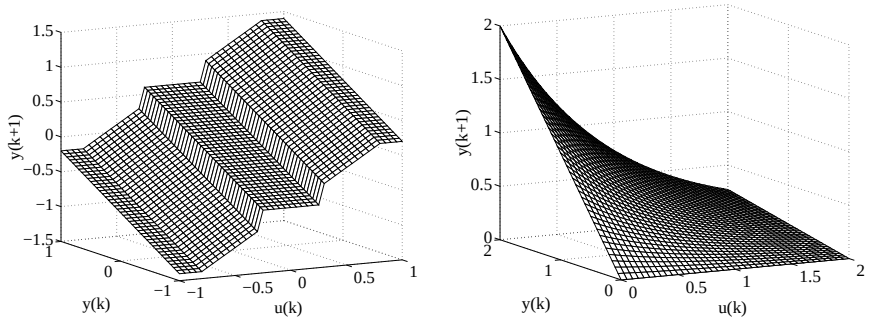
$$w = \begin{cases} 0, & -0.3 \le u \le 0.3, \\ u, & 0.3 \le |u| \le 0.8, \\ 0.8\,\mathrm{sign}(u), & 0.8 \le |u|. \end{cases} \tag{5.11b}$$

The input-output description of the system using the NARX model (3.58) can be seen as a surface in the space $(U \times Y \times Y) \subset \mathbb{R}^3$, as shown in Figure 5.7(a). As another example, consider a state-space system (Chen and Billings, 1989):

$$\begin{aligned} x(k+1) &= x(k) + u(k), \\ y(k) &= \exp(-x(k))\,. \end{aligned} \tag{5.12}$$

For this system, an input–output regression model $y(k+1) = y(k)\exp(-u(k))$ can be derived. The corresponding regression surface is shown in Figure 5.7(b). Note that if the measurements of the state of this system are available, the state and output mappings in (5.12) can be approximated separately, yielding one two-variate linear and one univariate nonlinear problem, which can be solved more easily.

□



(a) System with a dead zone and saturation.

(b) System $y(k+1) = y(k)\exp(-u(k))$.

Figure 5.7.: Regression surfaces of two nonlinear dynamic systems.

Example 5.4 (Identification of an Autoregressive System) Consider a time series generated by a nonlinear autoregressive system defined by (Ikoma and Hirota, 1993):

$$y(k+1) = f(y(k)) + \epsilon(k), \quad f(y) = \begin{cases} 2y - 2, & 0.5 \leq y \\ -2y, & -0.5 < y < 0.5 \\ 2y + 2, & y \leq -0.5 \end{cases} \tag{5.13}$$

Here, $\epsilon(k)$ is an independent random variable of $N(0, \sigma^2)$ with $\sigma = 0.3$. From the generated data $x(k)$ $k = 0, \ldots, 200$, with an initial condition $x(0) = 0.1$, the first 100 points are used for identification and the rest for model validation. By means of fuzzy clustering, a TS affine model with three reference fuzzy sets will be obtained. It is assumed that the only prior knowledge is that the data was generated by a nonlinear autoregressive system:

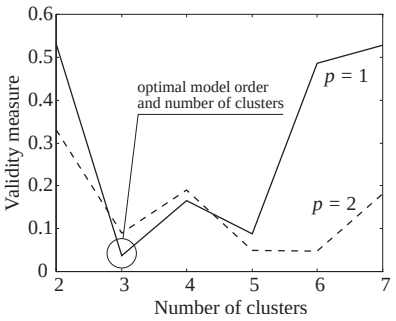
$$y(k+1) = f\big(y(k), y(k-1), \ldots, y(k-p+1)\big) = f\big(\mathbf{x}(k)\big), \tag{5.14}$$

where p is the system's order. Here $\mathbf{x}(k) = [y(k), y(k-1), \ldots, y(k-p+1)]^T$ is the regression vector and $y(k+1)$ is the response variable. The matrix $\mathbf{Z}$ is constructed from the identification data:

$$\mathbf{Z} = \begin{bmatrix} y(p) & y(p+1) & \cdots & y(N-1) \\ \cdots & \cdots & \cdots & \cdots \\ y(1) & y(2) & \cdots & y(N-p) \\ y(p+1) & y(p+2) & \cdots & y(N) \end{bmatrix}. \tag{5.15}$$

To identify the system we need to find the order p and to approximate the function f by a TS affine model. The order of the system and the number of clusters can be determined by means of a cluster validity measure which attains low values for "good" partitions (Babuška,

1998). This validity measure was calculated for a range of model orders $p = 1, 2 \ldots, 5$ and number of clusters $c = 2, 3 \ldots, 7$. The results are shown in a matrix form in Figure 5.8(b). The optimum (printed in boldface) was obtained for $p = 1$ and $c = 3$ which corresponds to (5.13). In Figure 5.8(a) the validity measure is plotted as a function of c for orders $p = 1, 2$. Note that this function may have several local minima, of which the first is usually chosen in order to obtain a simple model with few rules.



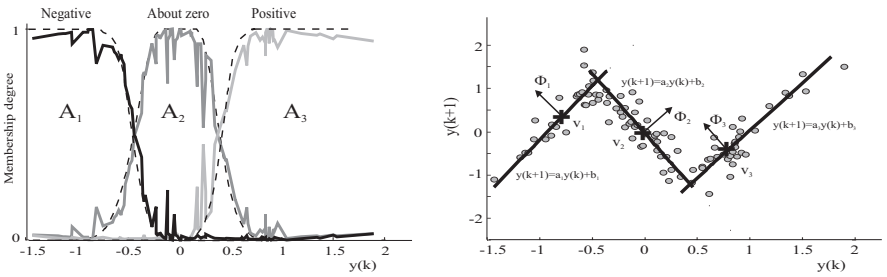
(a)

number of clusters V	model order 1	2	3	4	5
2	0.53	0.33	0.50	4.64	1.27
3	**0.03**	0.08	0.07	0.21	2.45
4	0.16	0.19	5.62	0.36	1.60
5	0.08	0.04	0.06	0.18	0.27
6	0.48	0.04	0.43	0.22	0.51
7	0.52	0.18	2.07	0.12	0.13

(b)

Figure 5.8.: The validity measure for different model orders and different number of clusters.

Figure 5.9(a) shows the projection of the obtained clusters onto the variable $y(k)$ for the correct system order $p = 1$ and the number of clusters $c = 3$.



(a) Fuzzy partition projected onto $y(k)$.

(b) Local linear models extracted from the clusters.

Figure 5.9.: Result of fuzzy clustering for $p = 1$ and $c = 3$. Part (a) shows the membership functions obtained by projecting the partition matrix onto $y(k)$. Part (b) gives the cluster prototypes v_i, the orientation of the eigenvectors Φ_i and the direction of the affine consequent models (lines).

Figure 5.9(b) shows also the cluster prototypes:

$$\mathbf{V} = \begin{bmatrix} -0.772 & -0.019 & 0.751 \\ 0.405 & 0.098 & -0.410 \end{bmatrix}.$$

From the cluster covariance matrices given below one can already see that the variance in one direction is higher than in the other one, thus the hyperellipsoids are flat and the model can be expected to represent a functional relationship between the variables involved

in clustering:

$$\mathbf{F}_1 = \begin{bmatrix} 0.057 & 0.099 \\ 0.099 & 0.249 \end{bmatrix}, \quad \mathbf{F}_2 = \begin{bmatrix} 0.063 & -0.099 \\ -0.099 & 0.224 \end{bmatrix}, \quad \mathbf{F}_3 = \begin{bmatrix} 0.065 & 0.107 \\ 0.107 & 0.261 \end{bmatrix}.$$

This is confirmed by examining the eigenvalues of the covariance matrices:

$$\begin{aligned} \lambda_{1,1} &= 0.015, & \lambda_{1,2} &= 0.291, \\ \lambda_{2,1} &= 0.017, & \lambda_{2,2} &= 0.271, \\ \lambda_{3,1} &= 0.018, & \lambda_{3,2} &= 0.308. \end{aligned}$$

One can see that for each cluster one of the eigenvalues is an order of magnitude smaller that the other one. By using least-squares estimation, we derive the parameters a_i and b_i of the affine TS model shown below. Piecewise exponential membership functions (2.13) are used to define the antecedent fuzzy sets. These functions were fitted to the projected clusters A_1 to A_3 by numerically optimizing the parameters c_l, c_r, w_l and w_r. The result is shown by dashed lines in Figure 5.9(a). After labeling these fuzzy sets NEGATIVE, ABOUT ZERO and POSITIVE, the obtained TS models can be written as:

$$\begin{aligned} &\text{If } y(k) \text{ is NEGATIVE} && \text{then } y(k+1) = 2.371y(k) + 1.237 \\ &\text{If } y(k) \text{ is ABOUT ZERO} && \text{then } y(k+1) = -2.109y(k) + 0.057 \\ &\text{If } y(k) \text{ is POSITIVE} && \text{then } y(k+1) = 2.267y(k) - 2.112 \end{aligned}$$

The estimated consequent parameters correspond approximately to the definition of the line segments in the deterministic part of (5.13). Also the partition of the antecedent domain is in agreement with the definition of the system.

□

5.4. Semi-Mechanistic Modeling

With physical insight in the system, nonlinear transformations of the measured signals can be involved. When modeling, for instance, the relation between the room temperature and the voltage applied to an electric heater, the power signal is computed by squaring the voltage, since it is the heater power rather than the voltage that causes the temperature to change (Lindskog and Ljung, 1994). This new variable is then used in a linear black-box model instead of the voltage itself. The motivation for using nonlinear regressors in nonlinear models is not to waste effort (rules, parameters, etc.) on estimating facts that are already known.

Another approach is based on a combination of white-box and black-box models. In many systems, such as chemical and biochemical processes, the modeling task can be divided into two subtasks: modeling of well-understood mechanisms based on mass and energy balances (first-principle modeling), and approximation of partially known relationships such as specific reaction rates. A number of hybrid modeling approaches have been proposed that combine first principles with nonlinear black-box models, e.g., neural networks (Psichogios and Ungar, 1992; Thompson and Kramer, 1994) or fuzzy models (Babuška et al., 1999). A neural network or a fuzzy model is typically used as a general nonlinear function

approximator that "learns" the unknown relationships from data and serves as a predictor of unmeasured process quantities that are difficult to model from first principles.

As an example, consider the modeling of a fed-batch stirred bioreactor described by the following equations derived from the mass balances (Psichogios and Ungar, 1992):

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \eta(\cdot)X - \frac{F}{V}X \tag{5.16a}$$

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -k_1\eta(\cdot)X + \frac{F}{V}[S_i - S] \tag{5.16b}$$

$$\frac{\mathrm{d}V}{\mathrm{d}t} = F \tag{5.16c}$$

where X is the biomass concentration, S is the substrate concentration, V is the reactor's volume, F is the inlet flow rate, k_1 is the substrate to cell conversion coefficient, and S_i is the inlet feed concentration. These mass balances provide a partial model. The kinetics of the process are represented by the specific growth rate $\eta(\cdot)$ which accounts for the conversion of the substrate to biomass, and it is typically a complex nonlinear function of the process variables. Many different models have been proposed to describe this function, but choosing the right model for a given process may not be straightforward. The hybrid approach is based on an approximation of $\eta(\cdot)$ by a nonlinear (black-box) model from process measurements and incorporates the identified nonlinear relation in the white-box model. The data can be obtained from batch experiments, for which $F = 0$, and (5.16a) reduces to the expression:

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \eta(\cdot)X, \tag{5.17}$$

where $\eta(\cdot)$ appears explicitly. This model is then used in the white-box model given by (5.16) for both batch and fed-batch regimes. An example of an application of the semi-mechanistic approach is the modeling of enzymatic Penicillin G conversion (Babuška et al., 1999), see Figure 5.10.

5.5. Summary and Concluding Remarks

Fuzzy modeling is a framework in which different modeling and identification methods are combined, providing, on the one hand, a transparent interface with the designer or the operator and, on the other hand, a flexible tool for nonlinear system modeling and control. The rule-based character of fuzzy models allows for a model interpretation in a way that is similar to the one humans use to describe reality. Conventional methods for statistical validation based on numerical data can be complemented by the human expertise, that often involves heuristic knowledge and intuition.

5.6. Problems

1. Explain the steps one should follow when designing a knowledge-based fuzzy model. One of the strengths of fuzzy systems is their ability to integrate prior knowledge and data. Explain how this can be done.

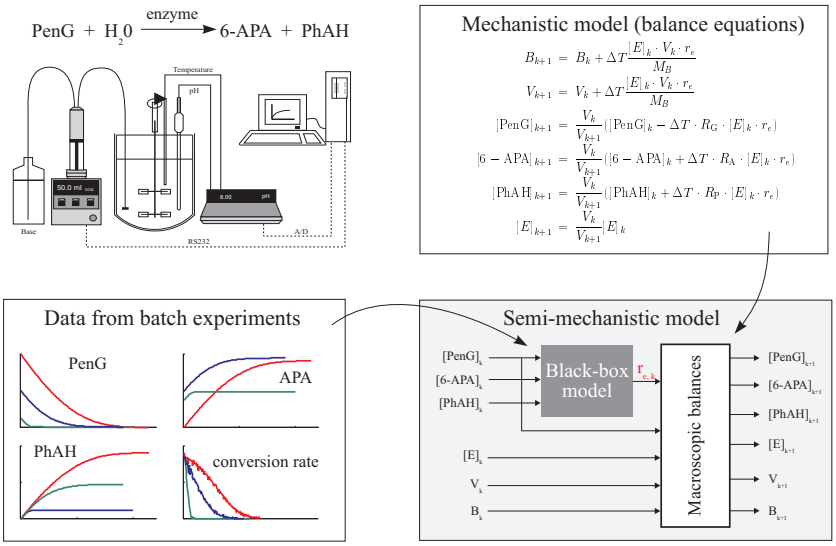


Figure 5.10.: Application of the semi-mechanistic modeling approach to a Penicillin G conversion process.

2. Consider a singleton fuzzy model $y = f(x)$ with the following two rules:

   1) **If** x is *Small* **then** $y = b_1$, 2) **If** x is *Large* **then** $y = b_2$.

   and membership functions as given in Figure 5.11.

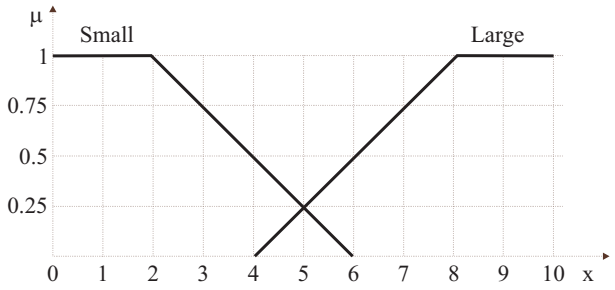


Figure 5.11.: Membership functions.

   Furthermore, the following data set is given:

   $x_1 = 1, \quad y_1 = 3$
   $x_2 = 5, \quad y_2 = 4.5$

   Compute the consequent parameters b_1 and b_2 such that the model gives the least summed squared error on the above data. What is the value of this summed squared error?

3. Consider the following fuzzy rules with singleton consequents:

   1) **If** x is A_1 **and** y is B_1 **then** $z = c_1$, 3) **If** x is A_1 **and** y is B_2 **then** $z = c_3$,
   2) **If** x is A_2 **and** y is B_1 **then** $z = c_2$, 4) **If** x is A_2 **and** y is B_2 **then** $z = c_4$.

Draw a scheme of the corresponding neuro-fuzzy network. What are the free (adjustable parameters in this network? What methods can be used to optimize these parameters by using input–output data?

4. Give a general equation for a NARX (Nonlinear AutoRegressive with eXogenous input) model. Explain all symbols. Give an example of a some NARX model of your choice.

5. Explain the term semi-mechanistic (hybrid) modeling. What do you understand under the terms "structure selection" and "parameter estimation" in case of such a model?

6. Knowledge-Based Fuzzy Control

The principles of knowledge-based fuzzy control are presented along with an overview of the basic fuzzy control schemes. Emphasis is put on the heuristic design of fuzzy controllers. Model-based design is addressed in Chapter 8.

Automatic control belongs to the application areas of fuzzy set theory that have attracted most attention. In 1974, the first successful application of fuzzy logic to control was reported (Mamdani, 1974). Control of cement kilns was an early industrial application (Holmblad and Østergaard, 1982). Since the first consumer product using fuzzy logic was marketed in 1987, the use of fuzzy control has increased substantially. A number of CAD environments for fuzzy control design have emerged together with VLSI hardware for fast execution. Fuzzy control is being applied to various systems in the process industry (Froese, 1993; Santhanam and Langari, 1994; Tani et al., 1994), consumer electronics (Hirota, 1993; Bonissone, 1994), automatic train operation (Yasunobu and Miyamoto, 1985) and traffic systems in general (Hellendoorn, 1993), and in many other fields (Hirota, 1993; Terano et al., 1994).

In this chapter, first the motivation for fuzzy control is given. Then, different fuzzy control concepts are explained: Mamdani, Takagi–Sugeno and supervisory fuzzy control. Finally, software and hardware tools for the design and implementation of fuzzy controllers are briefly addressed.

6.1. Motivation for Fuzzy Control

Conventional control theory uses a mathematical model of a process to be controlled and specifications of the desired closed-loop behaviour to design a controller. This approach may fall short if the model of the process is difficult to obtain, (partly) unknown, or highly nonlinear. The design of controllers for seemingly easy everyday tasks such as driving a car or grasping a fragile object continues to be a challenge for robotics, while these tasks are easily performed by human beings. Yet, humans do not use mathematical models nor exact trajectories for controlling such processes.

Many processes controlled by human operators in industry cannot be automated using conventional control techniques, since the performance of these controllers is often inferior to that of the operators. One of the reasons is that linear controllers, which are commonly used in conventional control, are not appropriate for nonlinear plants. Another reason is that humans aggregate various kinds of information and combine control strategies, that cannot be integrated into a single analytic control law. The underlying principle of *knowledge-based (expert) control* is to capture and implement experience and knowledge available from experts (e.g., process operators). A specific type of knowledge-based control is the fuzzy rule-based control, where the control actions corresponding to particular conditions of the system are described in terms of fuzzy if-then rules. Fuzzy sets are used to define the

meaning of qualitative values of the controller inputs and outputs such *small* error, *large* control action.

The early work in fuzzy control was motivated by a desire to

- mimic the control actions of an experienced human operator (knowledge-based part)
- obtain smooth interpolation between discrete outputs that would normally be obtained (fuzzy logic part)

Since then the application range of fuzzy control has widened substantially. However, the two main motivations still persevere. The linguistic nature of fuzzy control makes it possible to express process knowledge concerning how the process should be controlled or how the process behaves. The interpolation aspect of fuzzy control has led to the viewpoint where fuzzy systems are seen as smooth function approximation schemes.

In most cases a fuzzy controller is used for direct feedback control. However, it can also be used on the supervisory level as, e.g., a self-tuning device in a conventional PID controller. Also, fuzzy control is no longer only used to directly express a priori process knowledge. For example, a fuzzy controller can be derived from a fuzzy model obtained through system identification. Therefore, only a very general definition of fuzzy control can be given:

Definition 6.1 (Fuzzy Controller) *A* fuzzy controller *is a controller that contains a (nonlinear) mapping that has been defined by using fuzzy if-then rules.*

6.2. Fuzzy Control as a Parameterization of Controller's Nonlinearities

The key issues in the above definition are the *nonlinear mapping* and the fuzzy *if-then rules*. Increased industrial demands on quality and performance over a wide range of operating regions have led to an increased interest in nonlinear control methods during recent years. The advent of 'new' techniques such as fuzzy control, neural networks, wavelets, and hybrid systems has amplified the interest.

Nonlinear control is considered, e.g., when the process that should be controlled is nonlinear and/or when the performance specifications are nonlinear. Basically all real processes are nonlinear, either through nonlinear dynamics or through constraints on states, inputs and other variables. Two basic approaches can be followed:

- *Design through nonlinear modeling.* Nonlinear techniques can be used for process modeling. The derived process model can serve as the basis for model-based control design. The model may be used off-line during the design or on-line, as a part of the controller (see Chapter 8).
- *Model-free nonlinear control.* Nonlinear techniques can also be used to design the controller directly, without any process model. Nonlinear elements can be used in the feedback or in the feedforward path. In practice the nonlinear elements are often combined with linear filters.

A variety of methods can be used to define nonlinearities. They include analytical equations, fuzzy systems, sigmoidal neural networks, splines, radial basis functions, wavelets, locally linear models/controllers, discrete switching logic, lookup tables, etc. These methods represent different ways of parameterizing nonlinearities, see Figure 6.1.

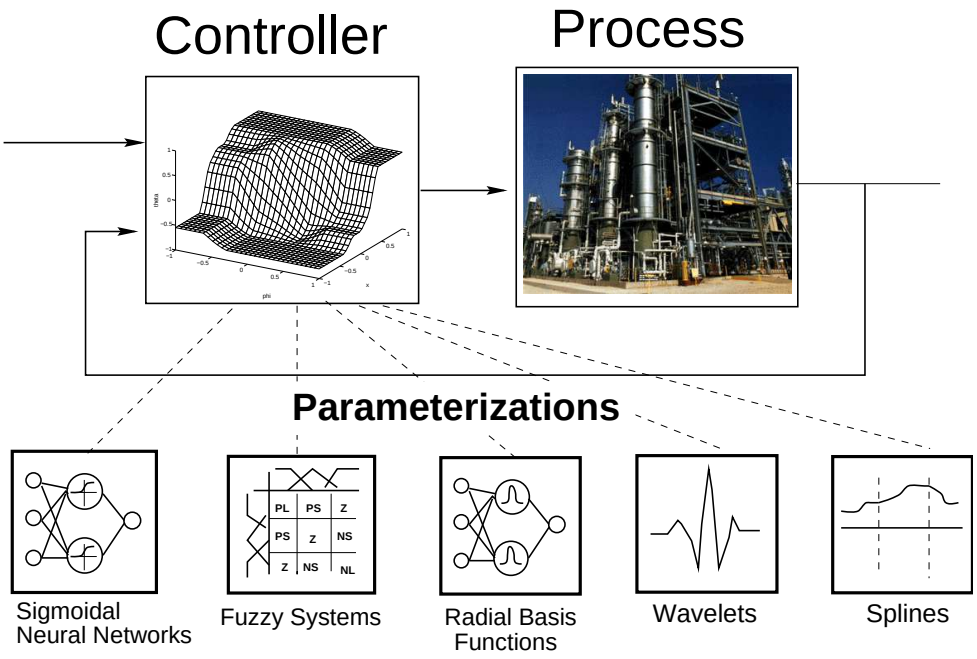


Figure 6.1.: Different parameterizations of nonlinear controllers.

Many of these methods have been shown to be universal function approximators for certain classes of functions. This means that they are capable of approximating a large class of functions are thus equivalent with respect to which nonlinearities that they can generate. Hence, it is of little value to argue whether one of the methods is better than the others if one considers only the closed loop control behavior. From the process' point of view it is the nonlinearity that matters and not how the nonlinearity is parameterized.

However, besides the approximation properties there are other important issues to consider. One of them is the *efficiency* of the approximation method in terms of the number of parameters needed to approximate a given function. Of great practical importance is whether the methods are local or global. Local methods allow for local adjustments. Examples of local methods are radial basis functions, splines, and fuzzy systems. How well the methods support the generation of nonlinearities from input/output data, i.e., identification/learning/training, is also of large interest. Another important issue is the availability of analysis and synthesis methods; how transparent the methods are, i.e., how readable the methods are and how easy it is to express prior process knowledge; the computational efficiency of the method; the availability of computer tools; and finally, subjective preferences such as how comfortable the designer/operator is with the method, and the level of training needed to use and understand the method.

Fuzzy logic systems appear to be favorable with respect to most of these criteria. They are universal approximators and, if certain design choices are made, the approximation is reasonably efficient. Depending on how the membership functions are defined the method

can be either global or local. It has similar estimation properties as, e.g., sigmoidal neural networks. Fuzzy logic systems can be very transparent and thereby they make it possible to express prior process knowledge well. A number of computer tools are available for fuzzy control implementation.

Fuzzy control can thus be regarded from two viewpoints. The first one focuses on the fuzzy if-then rules that are used to locally define the nonlinear mapping and can be seen as the user interface part of fuzzy systems. The second view consists of the nonlinear mapping that is generated from the rules and the inference process (Figure 6.2).

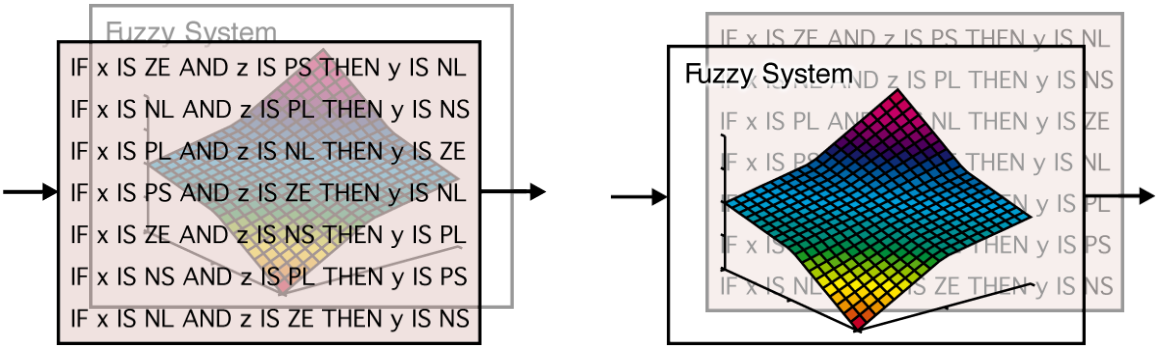


Figure 6.2.: The views of fuzzy systems. Fuzzy rules (left) are the user interface to the fuzzy system. They define a nonlinear mapping (right) which is the eventual input–output representation of the system.

The rules and the corresponding reasoning mechanism of a fuzzy controller can be of the different types introduced in Chapter 3. Most often used are

- *Mamdani (linguistic) controller* with either fuzzy or singleton consequents. This type of controller is usually used as a *direct* closed-loop controller.
- *Takagi–Sugeno (TS) controller*, typically used as a *supervisory* controller.

These two controllers are described in the following sections.

6.3. Mamdani Controller

Mamdani controller is usually used as a feedback controller. Since the rule base represents a static mapping between the antecedent and the consequent variables, external dynamic filters must be used to obtain the desired dynamic behavior of the controller (Figure 6.3).

The control protocol is stored in the form of if–then rules in a rule base which is a part of the *knowledge base*. While the rules are based on qualitative knowledge, the membership functions defining the linguistic terms provide a smooth interface to the numerical process variables and the set-points. The *fuzzifier* determines the membership degrees of the controller input values in the antecedent fuzzy sets. The inference mechanism combines this information with the knowledge stored in the rules and determines what the output of the rule-based system should be. In general, this output is again a fuzzy set. For control purposes, a crisp control signal is required. The *defuzzifier* calculates the value of this crisp signal from the fuzzy controller outputs.

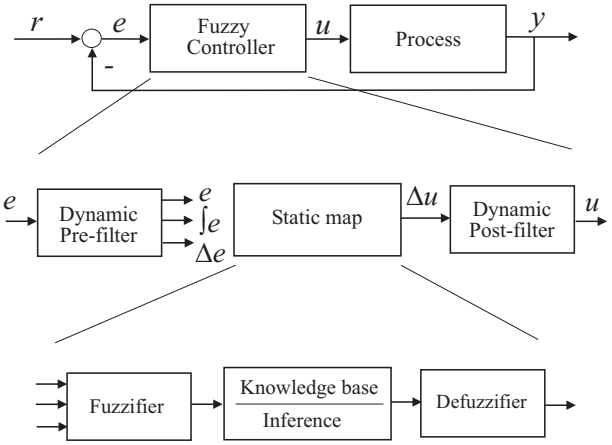


Figure 6.3.: Fuzzy controller in a closed-loop configuration (top panel) consists of dynamic filters and a static map (middle panel). The static map is formed by the knowledge base, inference mechanism and fuzzification and defuzzification interfaces.

From Figure 6.3 one can see that the fuzzy mapping is just one part of the fuzzy controller. Signal processing is required both before and after the fuzzy mapping.

6.3.1. Dynamic Pre-Filters

The *pre-filter* processes the controller's inputs in order to obtain the inputs of the static fuzzy system. It will typically perform some of the following operations on the input signals:

Signal Scaling

It is often convenient to work with signals on some normalized domain, e.g., $[-1, 1]$. This is accomplished by normalization gains which scale the input into the normalized domain $[-1, 1]$. Values that fall outside the normalized domain are mapped onto the appropriate endpoint.

Dynamic Filtering

In a fuzzy PID controller, for instance, linear filters are used to obtain the derivative and the integral of the control error e. Nonlinear filters are found in nonlinear observers, and in adaptive fuzzy control where they are used to obtain the fuzzy system parameter estimates.

Feature Extraction

Through the extraction of different features numeric transformations of the controller inputs are performed. These transformations may be Fourier or wavelet transforms, coordinate transformations or other basic operations performed on the fuzzy controller inputs.

6.3.2. Dynamic Post-Filters

The *post-filter* represents the signal processing performed on the fuzzy system's output to obtain the actual control signal. Operations that the post-filter may perform include:

Signal Scaling

A denormalization gain can be used which scales the output of the fuzzy system to the physical domain of the actuator signal.

Dynamic Filtering

In some cases, the output of the fuzzy system is the increment of the control action. The actual control signal is then obtained by integrating the control increments. Of course, other forms of smoothing devices and even nonlinear filters may be considered.

This decomposition of a controller to a static map and dynamic filters can be done for most classical control structures. To see this, consider a PID (Proportional-Integral-Differential) described by the following equation:

$$u(t) = Pe(t) + I\int_0^t e(\tau)\mathrm{d}\tau + D\frac{\mathrm{d}e(t)}{\mathrm{d}t}, \tag{6.1}$$

where $u(t)$ is the control signal fed to the process to be controlled and $e(t) = r(t) - y(t)$ is the error signal: the difference between the desired and measured process output. A computer implementation of a PID controller can be expressed as a difference equation:

$$u_{\mathrm{PID}}[k] = u_{\mathrm{PID}}[k-1] + k_I e[k] + k_P \Delta e[k] + k_D \Delta^2 e[k] \tag{6.2}$$

with:

$$\begin{aligned} \Delta e[k] &= e[k] - e[k-1] \\ \Delta^2 e[k] &= \Delta e[k] - \Delta e[k-1] \end{aligned}$$

The discrete-time gains k_P, k_I and k_D are for a given sampling period derived from the continuous time gains P, I and D. Equation (6.1) is linear function (geometrically a hyperplane):

$$u = \sum_{i=1}^{3} a_i x_i, \tag{6.3}$$

where $x_1 = e(t)$, $x_2 = \int_0^t e(\tau)\mathrm{d}\tau$, $x_3 = \frac{\mathrm{d}e(t)}{dt}$ and the a_i parameters are the P, I and D gains. The linear form (6.3) can be generalized to a nonlinear function:

$$\mathbf{u} = f(\mathbf{x}) \tag{6.4}$$

In the case of a fuzzy logic controller, the nonlinear function f is represented by a fuzzy mapping. Clearly, fuzzy controllers analogous to linear P, PI, PD or PID controllers can be designed by using appropriate dynamic filters such as differentiators and integrators.

6.3.3. Rule Base

Mamdani fuzzy systems are quite close in nature to manual control. The controller is defined by specifying what the output should be for a number of different input signal combinations. Each input signal combination is represented as a rule of the following form:

$$\mathcal{R}_i\text{: } \textbf{If } x_1 \text{ is } A_{i1} \ldots \textbf{ and } x_n \text{ is } A_{in} \textbf{ then } u \text{ is } B_i, \qquad i = 1, 2, \ldots, K\,. \tag{6.5}$$

Also other logical connectives and operators may be used, e.g., *or* and *not*. In Mamdani fuzzy systems the antecedent and consequent fuzzy sets are often chosen to be triangular or Gaussian. It is also common that the input membership functions overlap in such a way that the membership values of the rule antecedents always sum up to one. In this case, and if the rule base is on conjunctive form, one can interpret each rule as defining the output value for one point in the input space. The input space point is the point obtained by taking the centers of the input fuzzy sets and the output value is the center of the output fuzzy set. The fuzzy reasoning results in smooth interpolation between the points in the input space, see Figure 6.4.

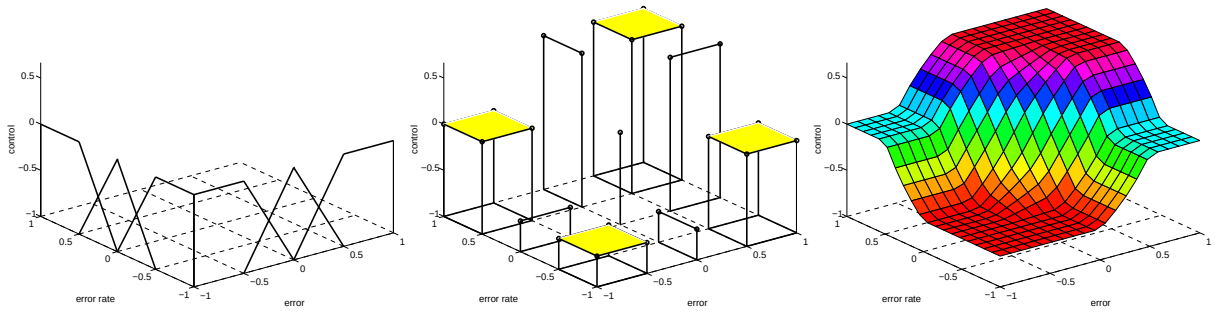


Figure 6.4.: Left: The membership functions partition the input space. Middle: Each rule defines the output value for one point or area in the input space. Right: The fuzzy logic interpolates between the constant values.

With this interpretation a Mamdani system can be viewed as defining a piecewise constant function with extensive interpolation. Depending on which inference methods that is used different interpolations are obtained. By proper choices it is even possible to obtain linear or multilinear interpolation. This is often achieved by replacing the consequent fuzzy sets by singletons. In such a case, inference and defuzzification are combined into one step, see Section 3.3, Equation (3.38).

Example 6.1 (Fuzzy PD Controller) Consider a fuzzy counterpart of a linear PD (proportional-derivative) controller. The rule base has two inputs – the error e, and the error change (derivative) $\dot{e}$, and one output – the control action u. An example of one possible rule base is:

		$\dot{e}$				
		NB	NS	ZE	PS	PB
e	NB	NB	NB	NS	NS	ZE
	NS	NB	NS	NS	ZE	PS
	ZE	NS	NS	ZE	PS	PS
	PS	NS	ZE	PS	PS	PB
	PB	ZE	PS	PS	PB	PB

Five linguistic terms are used for each variable, (NB – *Negative big*, NS – *Negative small*, ZE – *Zero*, PS – *Positive small* and PB – *Positive big*). Each entry of the table defines one rule, e.g. R_{23}: "**If** e is NS and $\dot{e}$ is ZE **then** u is NS". Figure 6.5 shows the resulting control surface obtained by plotting the inferred control action u for discretized values of e and $\dot{e}$.

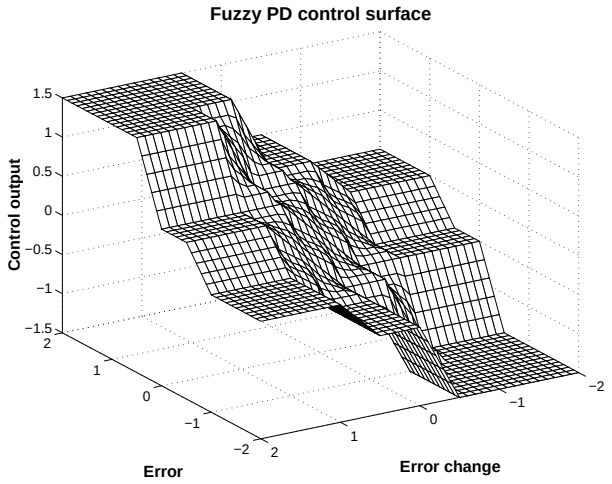


Figure 6.5.: Fuzzy PD control surface.

In fuzzy PD control, a simple difference $\Delta e = e(k) - e(k-1)$ is often used as a (poor) approximation for the derivative.

□

6.3.4. Design of a Fuzzy Controller

Determine Inputs and Outputs

In this step, one needs basic knowledge about the character of the process dynamics (stable, unstable, stationary, time-varying, etc.), the character of the nonlinearities, the control objectives and the constraints. The plant dynamics together with the control objectives determine the dynamics of the controller, e.g., a PI, PD or PID type fuzzy controller.

In order to compensate for the plant nonlinearities, time-varying behavior or other undesired phenomena, other variables than error and its derivative or integral may be used as the controller inputs. Typically, it can be the plant output(s), measured or reconstructed states, measured disturbances or other external variables. It is, however, important to realize that with an increasing number of inputs, the complexity of the fuzzy controller (i.e., the number of linguistic terms and the total number of rules) increases drastically.

For practical reasons, it is useful to recognize the influence of different variables and to decompose a fuzzy controller with many inputs into several simpler controllers with fewer inputs, working in parallel or in a hierarchical structure (see Section 3.2.7).

It is also important to realize that contrary to linear control, there is a difference between the incremental and absolute form of a fuzzy controller. An absolute form of a fuzzy PD controller, for instance, realizes a mapping $u = f(e, \dot{e})$, while its incremental form is a mapping $\dot{u} = f(\dot{e}, \ddot{e})$. With the incremental form, the possibly nonlinear control strategy relates to the rate of change of the control action while with the absolute form to the action itself. It has direct implications for the design of the rule base and also to some general properties of the controller. For instance, the output of a fuzzy controller in an absolute form is limited by definition, which is not true for the incremental form.

Another issue to consider is whether the fuzzy controller will be the first automatic controller in the particular application, or whether it will replace or complement an existing controller. In the latter case, the choice of the fuzzy controller structure may depend on the configuration of the current controller. Summarizing, we stress that this step is the most important one, since an inappropriately chosen structure can jeopardize the entire design, regardless of the rules or the membership functions.

Define Membership Functions and Scaling Factors

As shown in Figure 6.6, the linguistic terms, their membership functions and the domain scaling factors are a part of the fuzzy controller knowledge base.

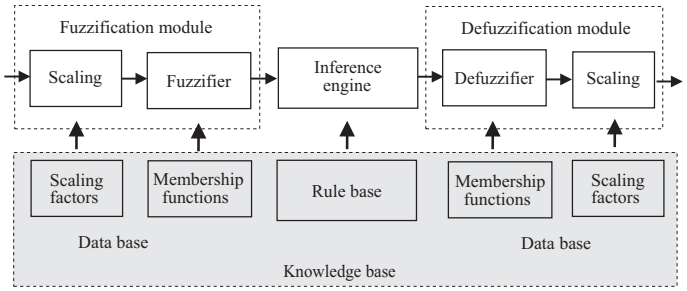


Figure 6.6.: Different modules of the fuzzy controller and the corresponding parts in the knowledge base.

First, the designer must decide, how many linguistic terms per input variable will be used. The number of rules needed for defining a complete rule base increases exponentially with the number of linguistic terms per input variable. In order to keep the rule base maintainable, the number of terms per variable should be low. On the other hand, with few terms, the flexibility in the rule base is restricted with respect to the achievable nonlinearity in the control mapping.

The number of terms should be carefully chosen, considering different settings for different variables according to their expected influence on the control strategy. A good choice may be to start with a few terms (e.g. 2 or 3 for the inputs and 5 for the outputs) and increase these numbers when needed. The linguistic terms have usually some meaning, i.e. they express magnitudes of some physical variables, such as *Small*, *Medium*, *Large*, etc. For

interval domains symmetrical around zero, the magnitude is combined with the sign, e.g. *Positive small* or *Negative medium*.

The membership functions may be a part of the expert's knowledge, e.g., the expert knows approximately what a "High temperature" means (in a particular application). If such knowledge is not available, membership functions of the same shape, uniformly distributed over the domain can be used as an initial setting and can be tuned later. For computational reasons, triangular and trapezoidal membership functions are usually preferred to bell-shaped functions.

Generally, the input and output variables are defined on restricted intervals of the real line. For simplification of the controller design, implementation and tuning, it is, however, more convenient to work with normalized domains, such as intervals $[-1, 1]$. Scaling factors are then used to transform the values from the operating ranges to these normalized domains. Scaling factors can be used for tuning the fuzzy controller gains too, similarly as with a PID.

Design the Rule Base

The construction of the rule base is a crucial aspect of the design, since the rule base encodes the control protocol of the fuzzy controller. Several methods of designing the rule base can be distinguished. One is based entirely on the expert's intuitive knowledge and experience. Since in practice it may be difficult to extract the control skills from the operators in a form suitable for constructing the rule base, this method is often combined with the control theory principles and a good understanding of the system's dynamics. Another approach uses a fuzzy model of the process from which the controller rule base is derived. Often, a "standard" rule base is used as a template. Such a rule base mimics the working of a linear controller of an appropriate type (for a PD controller has a typical form shown in Example 6.1. Notice that the rule base is symmetrical around its diagonal and corresponds to a linear form $u = Pe + D\dot{e}$. The gains P and D can be defined by a suitable choice of the scaling factors.

Tune the Controller

The tuning of a fuzzy controller is often compared to the tuning of a PID, stressing the large number of the fuzzy controller parameters, compared to the 3 gains of a PID. Two remarks are appropriate here. First, a fuzzy controller is a more general type of controller than a PID, capable of controlling nonlinear plants for which linear controller cannot be used directly, or improving the control of (almost) linear systems beyond the capabilities of linear controllers. For that, one has to pay by defining and tuning more controller parameters. Secondly, in case of complex plants, there is often a significant coupling among the effects of the three PID gains, and thus the tuning of a PID may be a very complex task. In fuzzy control, on the other hand, the rules and membership functions have local effects which is an advantage for control of nonlinear systems. For instance, non-symmetric control laws can be designed for systems exhibiting non-symmetric dynamic behaviour, such as thermal systems.

The scope of influence of the individual parameters of a fuzzy controller differs. The scaling factors, which determine the overall gain of the fuzzy controller and also the relative

gains of the individual controller inputs, have the most global effect. Notice, that changing a scaling factor also scales the possible nonlinearity defined in the rule base, which may not be desirable. The effect of the membership functions is more localized. A modification of a membership function, say *Small*, for a particular variable, influences only those rules, that use this term is used. Most local is the effect of the consequents of the individual rules. A change of a rule consequent influences only that region where the rule's antecedent holds.

As we already know, fuzzy inference systems are general function approximators, i.e. they can approximate any smooth function to any degree of accuracy. This means that a linear controller is a special case of a fuzzy controller, considered from the input–output functional point of view. Therefore, a fuzzy controller can be initialized by using an existing linear control law, which considerably simplifies the initial tuning phase while simultaneously guaranteeing a "minimal" performance of the fuzzy controller. The rule base or the membership functions can then be modified further in order to improve the system's performance or to eliminate influence of some (local) undesired phenomena like friction, etc. The following example demonstrates this approach.

Example 6.2 (Fuzzy Friction Compensation) In this example we will develop a fuzzy controller for a simulation of DC motor which includes a simplified model of static friction. This example is implemented in MATLAB/Simulink (`fricdemo.m`). Figure 6.7 shows a block diagram of the DC motor.

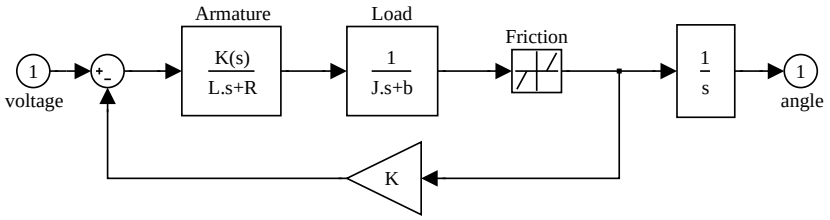


Figure 6.7.: DC motor with friction.

First, a linear proportional controller is designed by using standard methods (root locus, for instance). Then, a proportional fuzzy controller is developed that exactly mimics a linear controller. The two controllers have identical responses and both suffer from a steady state error due to the friction. Special rules are added to the rule bases in order to reduce this error. The linear and fuzzy controllers are compared by using the block diagram in Figure 6.8.

The fuzzy control rules that mimic the linear controller are:

```
If error is Zero
   then control input is Zero;
If error is Positive Big
   then control input is Positive Big;
If error is Negative Big
   then control input is Negative Big;
```

The control result achieved with this controller is shown in Figure 6.9.

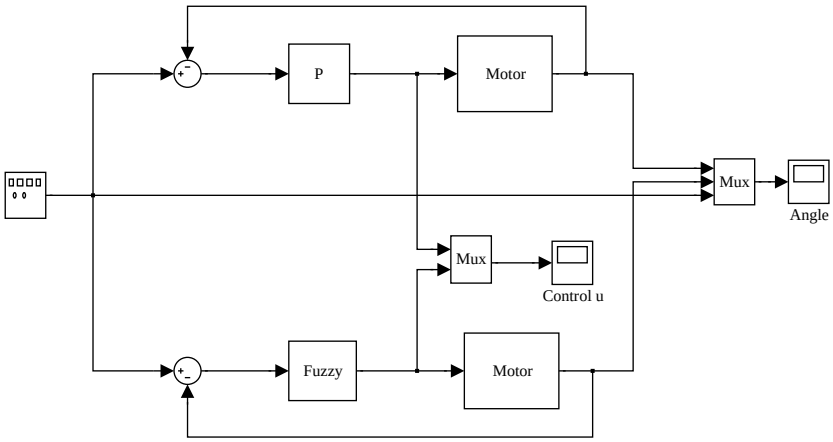


Figure 6.8.: Block diagram for the comparison of proportional linear and fuzzy controllers.

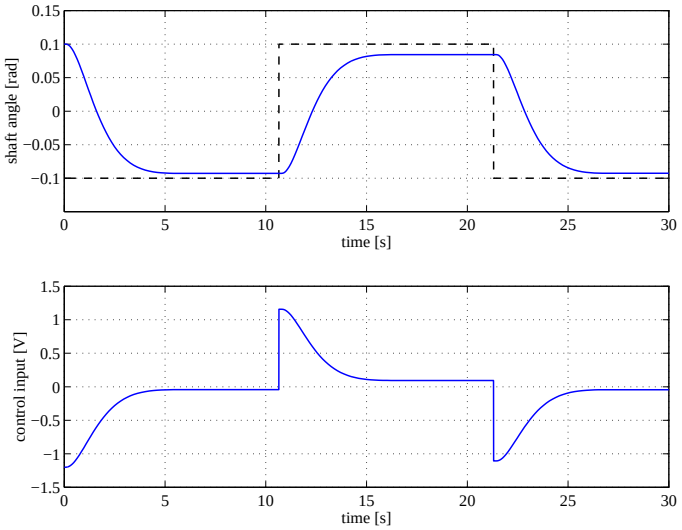


Figure 6.9.: Response of the linear controller to step changes in the desired angle.

Two additional rules are included to prevent the controller from generating a *small* control action whenever the control error is *small*. Such a control action obviously does not have any influence on the motor, as it is not able to overcome the friction.

```
If error is Negative Small
   then control input is NOT Negative Small;
If error is Positive Small
   then control input is NOT Positive Small;
```

Membership functions for the linguistic terms “Negative Small” and “Positive Small” have been derived from the result in Figure 6.9. Łukasiewicz implication is used in order to properly handle the *not* operator (see Example 3.7 for details). The control result achieved with this fuzzy controller is shown in Figure 6.10 Note that the steady-state error has almost been eliminated.

Other than fuzzy solutions to the friction problem include PI control and sliding-mode

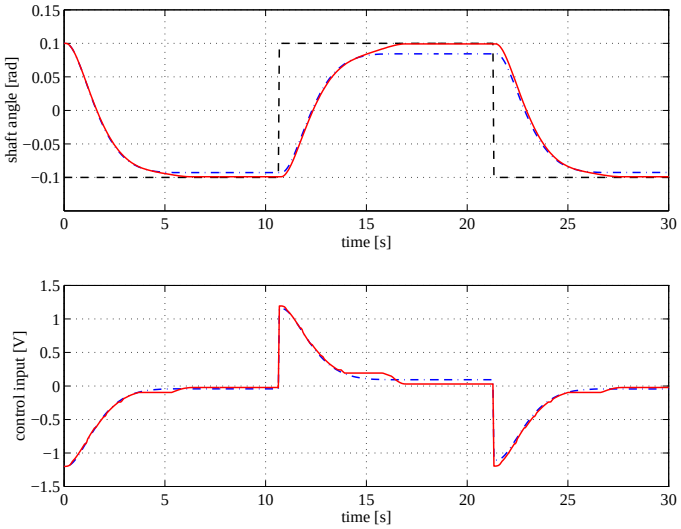


Figure 6.10.: Comparison of the linear controller (dashed-dotted line) and the fuzzy controller (solid line).

control. The integral action of the PI controller will introduce oscillations in the loop and thus deteriorate the control performance. The reason is that the friction nonlinearity introduces a discontinuity in the loop. The sliding-mode controller is robust with regard to nonlinearities in the process. It also reacts faster than the fuzzy controller, but at the cost of violent control actions (Figure 6.11).

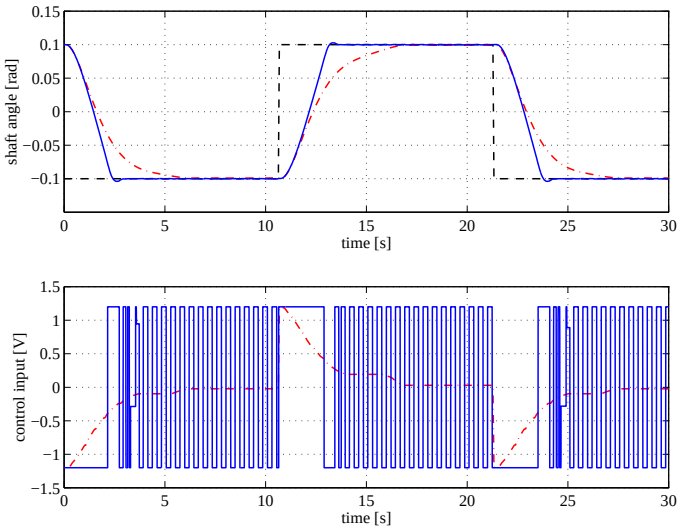


Figure 6.11.: Comparison of the fuzzy controller (dashed-dotted line) and a sliding-mode controller (solid line).

□

6.4. Takagi–Sugeno Controller

Takagi–Sugeno (TS) fuzzy controllers are close to gain scheduling approaches. Several linear controllers are defined, each valid in one particular region of the controller's input space. The total controller's output is obtained by selecting one of the controllers based on the value of the inputs (classical gain scheduling), or by interpolating between several of the linear controllers (fuzzy gain scheduling, TS control), see Figure 6.12.

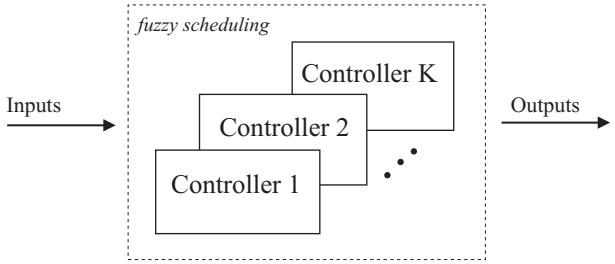


Figure 6.12.: The TS fuzzy controller can be seen as a collection of several local controllers combined by a fuzzy scheduling mechanism.

When TS fuzzy systems are used it is common that the input fuzzy sets are trapezoidal. Each fuzzy set determines a region in the input space where, in the linear case, the output is determined by a linear function of the inputs. Fuzzy logic is only used to interpolate in the cases where the regions in the input space overlap. Such a TS fuzzy system can be viewed as piecewise linear (affine) function with limited interpolation. An example of a TS control rule base is

$$\begin{aligned} \mathcal{R}_1 \quad &: \quad \textbf{If } r \text{ is } \textit{Low} \textbf{ then } u_1 = P_{\text{Low}} e + D_{\text{Low}} \dot{e} \\ \mathcal{R}_2 \quad &: \quad \textbf{If } r \text{ is } \textit{High} \textbf{ then } u_2 = P_{\text{High}} e + D_{\text{High}} \dot{e} \end{aligned} \tag{6.6}$$

Note here that the antecedent variable is the reference r while the consequent variables are the error e and its derivative $\dot{e}$. The controller is thus linear in e and $\dot{e}$, but the parameters of the linear mapping depend on the reference:

$$\begin{aligned} u &= \frac{\mu_{\text{Low}}(r) u_1 + \mu_{\text{High}}(r) u_2}{\mu_{\text{Low}}(r) + \mu_{\text{High}}(r)} \\ &= \frac{\mu_{\text{Low}}(r)\big(P_{\text{Low}} e + D_{\text{Low}} \dot{e}\big) + \mu_{\text{High}}(r)\big(P_{\text{High}} e + D_{\text{High}} \dot{e}\big)}{\mu_{\text{Low}}(r) + \mu_{\text{High}}(r)} \end{aligned} \tag{6.7}$$

If the local controllers differ only in their parameters, the TS controller is a rule-based form of a gain-scheduling mechanism. On the other hand, *heterogeneous control* (Kuipers and Aström, 1994) can employ different control laws in different operating regions. In the latter case, e.g. time-optimal control for dynamic transitions can be combined with PI(D) control in the vicinity of setpoints. Therefore, the TS controller can be seen as a simple form of supervisory control.

6.5. Fuzzy Supervisory Control

A fuzzy inference system can also be applied at a higher, supervisory level of the control hierarchy. A supervisory controller is a secondary controller which augments the existing

controller so that the control objectives can be met which would not be possible without the supervision. A supervisory controller can, for instance, adjust the parameters of a low-level controller according to the process information (Figure 6.13).

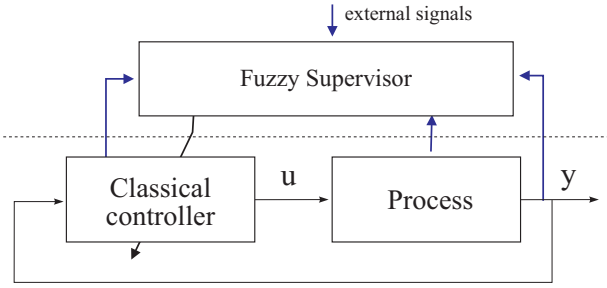


Figure 6.13.: Fuzzy supervisory control.

In this way, static or dynamic behavior of the low-level control system can be modified in order to cope with process nonlinearities or changes in the operating or environmental conditions. An advantage of a supervisory structure is that it can be added to already existing control systems. Hence, the original controllers can always be used as initial controllers for which the supervisory controller can be tuned for improving the performance. A supervisory structure can be used for implementing different control strategies in a single controller. An example is choosing proportional control with a high gain, when the system is very far from the desired reference signal and switching to a PI-control in the neighborhood of the reference signal. Because the parameters are changed during the dynamic response, supervisory controllers are in general nonlinear.

Many processes in the industry are controlled by PID controllers. Despite their advantages, conventional PID controllers suffer from the fact that the controller must be re-tuned when the operating conditions change. This disadvantage can be reduced by using a fuzzy supervisor for adjusting the parameters of the low-level controller. A set of rules can be obtained from experts to adjust the gains P and D of a PD controller, for example based on the current set-point r. The rules may look like:

If process output is *High*
then reduce proportional gain *Slightly* **and**
increase derivative gain *Moderately*.

The TS controller can be interpreted as a simple version of supervisory control. For instance, the TS rules (6.6) can be written in terms of Mamdani or singleton rules that have the P and D parameters as outputs. These are then passed to a standard PD controller at a lower level.

Example 6.3 A supervisory fuzzy controller has been applied to pressure control in a laboratory fermenter, depicted in Figure 6.14.

The volume of the fermenter tank is 40 l, and normally it is filled with 25 l of water. At the bottom of the tank, air is fed into the water at a constant flow-rate, kept constant by a local mass-flow controller. The air pressure above the water level is controlled by an outlet

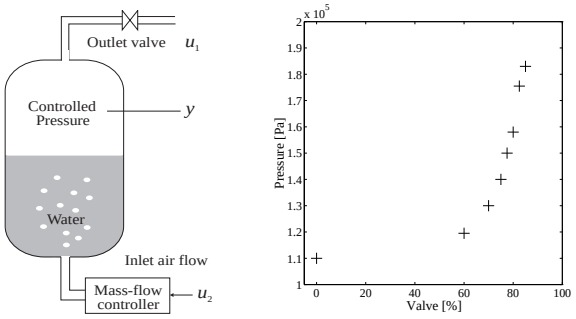


Figure 6.14.: Left: experimental setup; right: nonlinear steady-state characteristic.

valve at the top of the tank. With a constant input flow-rate, the system has a single input, the valve position, and a single output, the air pressure. Because of the underlying physical mechanisms, and because of the nonlinear characteristic of the control valve, the process has a nonlinear steady-state characteristic, shown in Figure 6.14, as well as a nonlinear dynamic behavior.

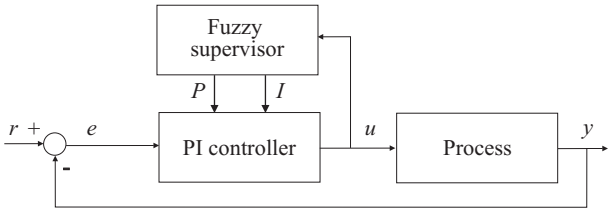


Figure 6.15.: The supervisory fuzzy control scheme.

A single-input, two-output supervisor shown in Figure 6.15 was designed. The input of the supervisor is the valve position $u(k)$ and the outputs are the proportional and the integral gain of a conventional PI controller. The supervisor updates the PI gains at each sample of the low-level control loop (5 s).

The domain of the valve position (0–100%) was partitioned into four fuzzy sets ('Small', 'Medium', 'Big' and 'Very Big'), see the membership functions in Figure 6.16.

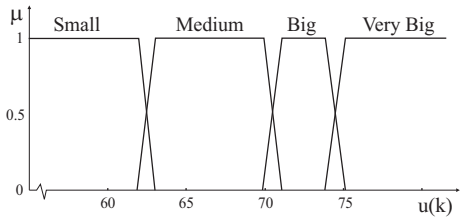


Figure 6.16.: Membership functions for $u(k)$.

The PI gains P and I associated with each of the fuzzy sets are given as follows:

Gains $\backslash u(k)$	Small	Medium	Big	Very big
P	190	170	155	140
I	150	90	70	50

The P and I values were found through simulations in the respective regions of the valve positions. The overall output of the supervisor is computed as a weighted mean of the local gains.

The supervisory fuzzy controller, tested and tuned through simulations, was applied to the process directly (without further tuning), under the nominal conditions. The real-time control results are shown in Figure 6.17.

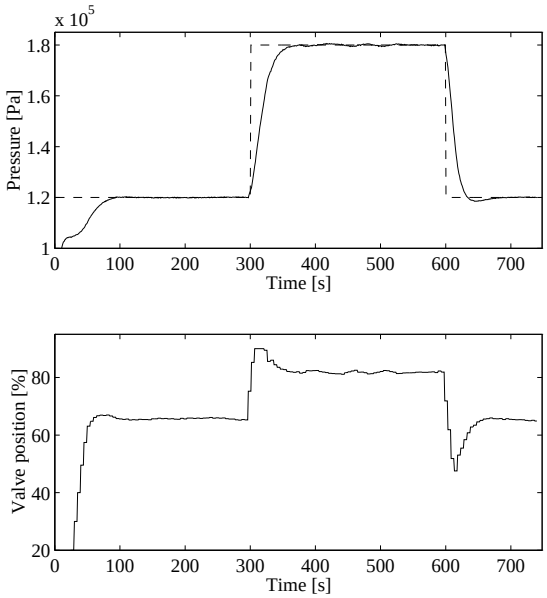


Figure 6.17.: Real-time control result of the supervisory fuzzy controller.

□

6.6. Operator Support

Despite all the advances in the automatic control theory, the degree of automation in many industries (such as chemical, biochemical or food industry) is quite low. Though basic automatic control loops are usually implemented, human operators must supervise and coordinate their function, set or tune the parameters and also control the process manually during the start-up, shut-down or transition phases. These types of control strategies cannot be represented in an analytical form but rather as if-then rules. By implementing the operator's expertise, the resulting fuzzy controller can be used as a decision support for advising less experienced operators (taking advantage of the transparent knowledge representation in the fuzzy controller). In this way, the variance in the quality of different operators is reduced, which leads to the reduction of energy and material costs, etc. The fuzzy system can simplify the operator's task by extracting relevant information from a large number of measurements and data. A suitable user interface needs to be designed for communication with the operators. The use of linguistic variables and a possible explanation facility in terms of these variables can improve the man–machine interface.

6.7. Software and Hardware Tools

Since the development of fuzzy controllers relies on intensive interaction with the designer, special software tools have been introduced by various software (SW) and hardware (HW) suppliers such as Omron, Siemens, Aptronix, Inform, National Semiconductors, etc. Most of the programs run on a PC, under Windows, some of them are available also for UNIX systems. See `http://www.isis.ecs.soton.ac.uk/resources/nfinfo/` for an extensive list.

Fuzzy control is also gradually becoming a standard option in plant-wide control systems, such as the systems from Honeywell. Most software tools consist of the following blocks.

6.7.1. Project Editor

The heart of the user interface is a graphical *project editor* that allows the user to build a fuzzy control system from basic blocks. Input and output variables can be defined and connected to the fuzzy inference unit either directly or via pre-processing or post-processing elements such as dynamic filters, integrators, differentiators, etc. The functions of these blocks are defined by the user, using the C language or its modification. Several fuzzy inference units can be combined to create more complicated (e.g., hierarchical or distributed) fuzzy control schemes.

6.7.2. Rule Base and Membership Functions

The rule base and the related fuzzy sets (membership functions) are defined using the rule base and membership function editors. The *rule base editor* is a spreadsheet or a table where the rules can be entered or modified. The *membership functions editor* is a graphical environment for defining the shape and position of the membership functions. Figure 6.18 gives an example of the various interface screens of FuzzyTech.

6.7.3. Analysis and Simulation Tools

After the rules and membership functions have been designed, the function of the fuzzy controller can be tested using tools for *static analysis* and *dynamic simulation*. Input values can be entered from the keyboard or read from a file in order to check whether the controller generates expected outputs. The degree of fulfillment of each rule, the adjusted output fuzzy sets, the results of rule aggregation and defuzzification can be displayed on line or logged in a file. For a selected pair of inputs and a selected output the control surface can be examined in two or three dimensions. Some packages also provide function for automatic checking of completeness and redundancy of the rules in the rule base. Dynamic behavior of the closed loop system can be analyzed in simulation, either directly in the design environment or by generating a code for an independent simulation program (e.g., Simulink).

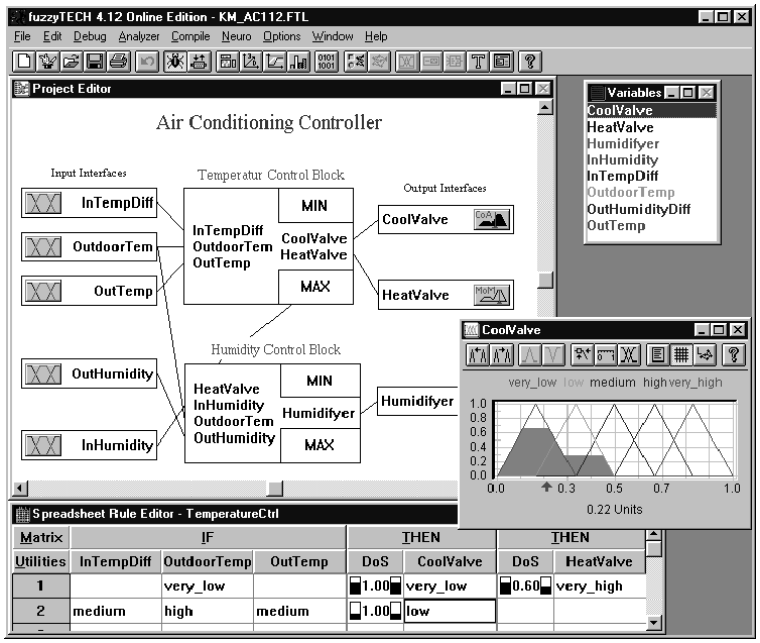


Figure 6.18.: Interface screens of FuzzyTech (Inform).

6.7.4. Code Generation and Communication Links

Once the fuzzy controller is tested using the software analysis tools, it can be used for controlling the plant either directly from the environment (via computer ports or analog inputs/outputs), or through generating a run-time code. Most of the programs generate a standard C-code and also a machine code for a specific hardware, such as microcontrollers or programmable logic controllers (PLCs). In this way, existing hardware can be also used for fuzzy control. Besides that, specialized fuzzy hardware is marketed, such as fuzzy control chips (both analog and digital, see Figure 6.19) or fuzzy coprocessors for PLCs.

Figure 6.19.: Fuzzy inference chip (Siemens).

6.8. Summary and Concluding Remarks

A fuzzy logic controller can be seen as a small real-time expert system implementing a part of human operator's or process engineer's expertise. From the control engineering perspective, a fuzzy controller is a nonlinear controller. In many implementations a PID-like controller is used, where the controller output is a function of the error signal and its derivatives. The applications in the industry are also increasing. Major producers of consumer goods use fuzzy logic controllers in their designs for consumer electronics, dishwashers, washing machines, automatic car transmission systems etc., even though this fact is not always advertised.

Fuzzy control is a new technique that should be seen as an extension to existing control methods and not their replacement. It provides an extra set of tools which the control engineer has to learn how to use where it makes sense. Nonlinear and partially known systems that pose problems to conventional control techniques can be tackled using fuzzy control. In this way, the control engineering is a step closer to achieving a higher level of automation in places where it has not been possible before.

In the academic world a large amount of research is devoted to fuzzy control. The focus is on analysis and synthesis methods. For certain classes of fuzzy systems, e.g., linear Takagi-Sugeno systems, many concepts results have been developed.

6.9. Problems

1. There are various ways to parameterize nonlinear models and controllers. Name at least three different parameterizations and explain how they differ from each other.

2. Draw a control scheme with a fuzzy PD (proportional-derivative) controller, including the process. Explain the internal structure of the fuzzy PD controller, including the dynamic filter(s), rule base, etc.

3. Give an example of a rule base and the corresponding membership functions for a fuzzy PI (proportional-integral) controller. What are the design parameters of this controller and how can you determine them?

4. State in your own words a definition of a fuzzy controller. How do fuzzy controllers differ from linear controllers, such as PID or state-feedback control? For what kinds of processes have fuzzy controllers the potential of providing better performance than linear controllers?

5. a) Give an example of several rules of a Takagi–Sugeno fuzzy controller. b) What are the design parameters of this controller? c) Give an example of a process to which you would apply this controller.

6. Is special fuzzy-logic hardware always needed to implement a fuzzy controller? Explain your answer.

7. Artificial Neural Networks

7.1. Introduction

Both neural networks and fuzzy systems are motivated by imitating human reasoning processes. In fuzzy systems, relationships are represented explicitly in the form of if–then rules. In neural networks, the relations are not explicitly given, but are "coded" in a network and its parameters. In contrast to knowledge-based techniques, no explicit knowledge is needed for the application of neural nets.

Artificial neural nets (ANNs) can be regarded as a functional imitation of biological neural networks and as such they share some advantages that biological organisms have over standard computational systems. The main feature of an ANN is its ability to learn complex functional relations by generalizing from a limited amount of training data. Neural nets can thus be used as (black-box) models of nonlinear, multivariable static and dynamic systems and can be trained by using input–output data observed on the system.

The research in ANNs started with attempts to model the biophysiology of the brain, creating models which would be capable of mimicking human thought processes on a computational or even hardware level. Humans are able to do complex tasks like perception, pattern recognition, or reasoning much more efficiently than state-of-the-art computers. They are also able to learn from examples and human neural systems are to some extent fault tolerant. These properties make ANN suitable candidates for various engineering applications such as pattern recognition, classification, function approximation, system identification, etc.

The most common ANNs consist of several layers of simple processing elements called neurons, interconnections among them and weights assigned to these interconnections. The information relevant to the input–output mapping of the net is stored in the weights.

7.2. Biological Neuron

A biological neuron consists of a *body* (or *soma*), an *axon* and a large number of *dendrites* (Figure 7.1). The dendrites are inputs of the neuron, while the axon is its output. The axon of a single neuron forms synaptic connections with many other neurons. It is a long, thin tube which splits into branches terminating in little bulbs touching the dendrites of other neurons. The small gap between such a bulb and a dendrite of another cell is called a *synapse*.

Impulses propagate down the axon of a neuron and impinge upon the synapses, sending signals of various strengths to the dendrites of other neurons. The strength of these signals is determined by the *efficiency* of the synaptic transmission. A signal acting upon a dendrite may be either *inhibitory* or *excitatory*. A biological neuron fires, i.e., sends an impulse down

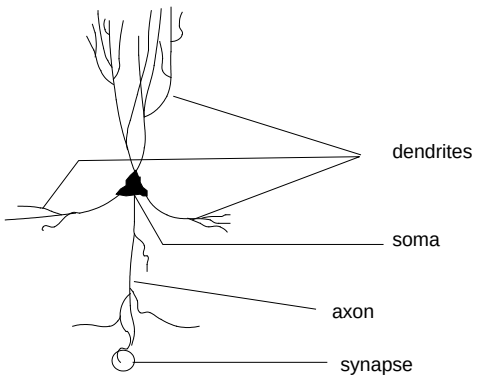


Figure 7.1.: Schematic representation of a biological neuron.

its axon, if the excitation level exceeds its inhibition by a critical amount, the *threshold* of the neuron.

Research into *models* of the human brain already started in the 19th century (James, 1890). It took until 1943 before McCulloch and Pitts (1943) formulated the first ideas in a mathematical model called the McCulloch-Pitts neuron. In 1957, a first multi-layer neural network model called the perceptron was proposed. However, significant progress in neural network research was only possible after the introduction of the backpropagation method (Rumelhart et al., 1986), which can train multi-layered networks.

7.3. Artificial Neuron

Mathematical models of biological neurons (called artificial neurons) mimic the functionality of biological neurons at various levels of detail. Here, a simple model will be considered, which is basically a static function with several inputs (representing the dendrites) and one output (the axon). Each input is associated with a weight factor (synaptic strength). The weighted inputs are added up and passed through a nonlinear function, which is called the *activation function*. The value of this function is the output of the neuron (Figure 7.2).

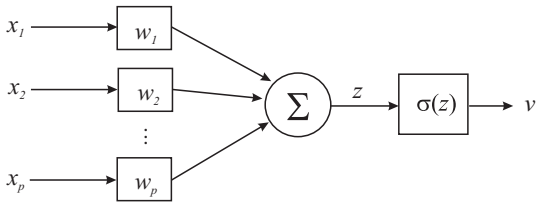


Figure 7.2.: Artificial neuron.

The weighted sum of the inputs is denoted by

$$z = \sum_{i=1}^{p} w_i x_i = \mathbf{w}^T \mathbf{x} \,. \tag{7.1}$$

Sometimes, a bias is added when computing the neuron's activation:

$$z = \sum_{i=1}^{p} w_i x_i + b = [\mathbf{w}^T b] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} .$$

This bias can be regarded as an extra weight from a constant (unity) input. To keep the notation simple, (7.1) will be used in the sequel.

The activation function maps the neuron's activation z into a certain interval, such as $[0, 1]$ or $[-1, 1]$. Often used activation functions are a threshold, sigmoidal and tangent hyperbolic functions (Figure 7.3).

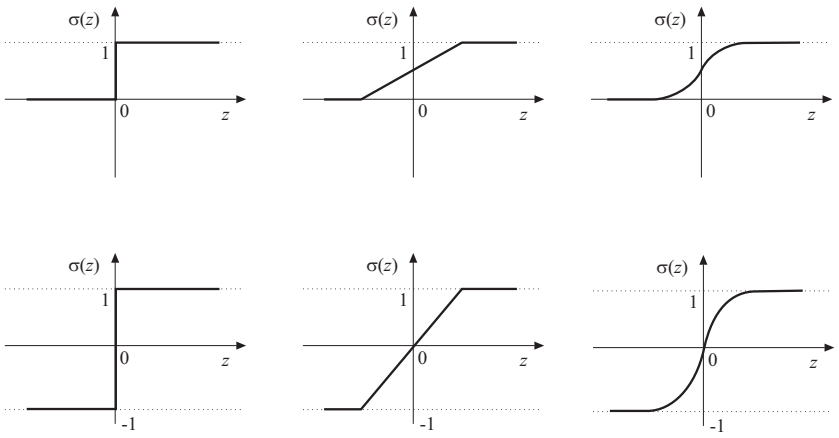


Figure 7.3.: Different types of activation functions.

- Threshold function (hard limiter)

$$\sigma(z) = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases} .$$

- Piece-wise linear (saturated) function

$$\sigma(z) = \begin{cases} 0 & \text{for } z < -\alpha \\ \frac{1}{2\alpha}(z + \alpha) & \text{for } -\alpha \leq z \leq \alpha \\ 1 & \text{for } z > \alpha \end{cases} .$$

- Sigmoidal function

$$\sigma(z) = \frac{1}{1 + \exp(-sz)}$$

  Here, s is a constant determining the steepness of the sigmoidal curve. For $s \to 0$ the sigmoid is very flat and for $s \to \infty$ it approaches a threshold function. Figure 7.4 shows three curves for different values of s. Typically $s = 1$ is used (the bold curve in Figure 7.4).

- Tangent hyperbolic

$$\sigma(z) = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}$$

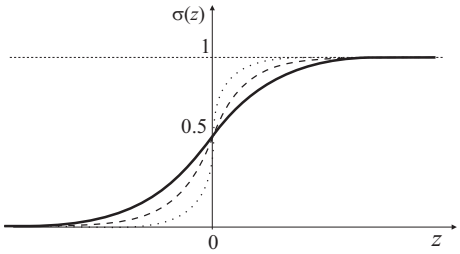


Figure 7.4.: Sigmoidal activation function.

7.4. Neural Network Architecture

Artificial neural networks consist of interconnected neurons. Neurons are usually arranged in several *layers*. This arrangement is referred to as the *architecture* of a neural net. Networks with several layers are called *multi-layer* networks, as opposed to *single-layer* networks that only have one layer. Within and among the layers, neurons can be interconnected in two basic ways:

- *Feedforward networks*: neurons are arranged in several layers. Information flows only in one direction, from the input layer to the output layer.
- *Recurrent networks*: neurons are arranged in one or more layers and feedback is introduced either internally in the neurons, to other neurons in the same layer or to neurons in preceding layers.

Figure 7.5 shows an example of a multi-layer feedforward ANN (perceptron) and a single-layer recurrent network (Hopfield network).

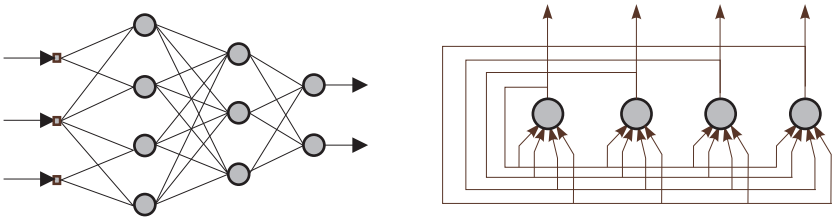


Figure 7.5.: Multi-layer feedforward ANN (left) and a single-layer recurrent (Hopfield) network (right).

In the sequel, we will concentrate on feedforward multi-layer neural networks and on a special single-layer architecture called the radial basis function network.

7.5. Learning

The learning process in biological neural networks is based on the change of the interconnection strength among neurons. Synaptic connections among neurons that simultaneously exhibit high activity are strengthened.

In artificial neural networks, various concepts are used. A mathematical approximation of biological learning, called Hebbian learning is used, for instance, in the Hopfield network.

Multi-layer nets, however, typically use some kind of optimization strategy whose aim is to minimize the difference between the desired and actual behavior (output) of the net.

Two different learning methods can be recognized: supervised and unsupervised learning:

- *Supervised learning*: the network is supplied with both the input values and the correct output values, and the weight adjustments performed by the network are based upon the error of the computed output. This method is presented in Section 7.6.3.
- *Unsupervised learning*: the network is only provided with the input values, and the weight adjustments are based only on the input values and the current network output. Unsupervised learning methods are quite similar to clustering approaches, see Chapter 4.

In the sequel, only supervised learning is considered.

7.6. Multi-Layer Neural Network

A multi-layer neural network (MNN) has one input layer, one output layer and an number of hidden layers between them (see Figure 7.6).

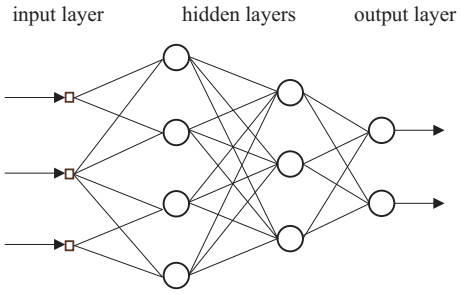


Figure 7.6.: A typical multi-layer neural network consists of an input layer, one or more hidden layers and an output layer.

A dynamic network can be realized by using a static feedforward network combined with a feedback connection. The output of the network is fed back to its input through delay operators z^{-1}. Figure 7.7 shows an example of a neural-net representation of a first-order system $y(k+1) = f_{\text{nn}}(y(k), u(k))$. In this way, a dynamic identification problem is reformulated as a static function-approximation problem (see Section 3.6 for a more general discussion).

In a MNN, two computational phases are distinguished:

1. *Feedforward computation.* From the network inputs x_i, $i = 1, \ldots, N$, the outputs of the first hidden layer are first computed. Then using these values as inputs to the second hidden layer, the outputs of this layer are computed, etc. Finally, the output of the network is obtained.
2. *Weight adaptation.* The output of the network is compared to the desired output. The difference of these two values called the error, is then used to adjust the weights first in the output layer, then in the layer before, etc., in order to decrease the error. This backward computation is called error backpropagation.

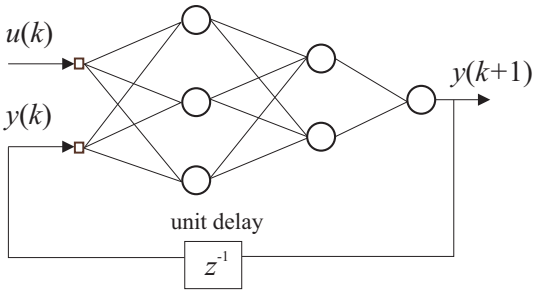


Figure 7.7.: A neural-network model of a first-order dynamic system.

The error backpropagation algorithm was proposed independently by Werbos (1974) and Rumelhart et al. (1986). The derivation of the algorithm is briefly presented in the following section.

7.6.1. Feedforward Computation

For simplicity, consider a MNN with one hidden layer (Figure 7.8). The input-layer neurons do not perform any computations, they merely distribute the inputs x_i to the weights w_{ij}^h of the hidden layer. The neurons in the hidden layer contain the *tanh* activation function, while the output neurons are usually linear. The output-layer weights are denoted by w_{ij}^o.

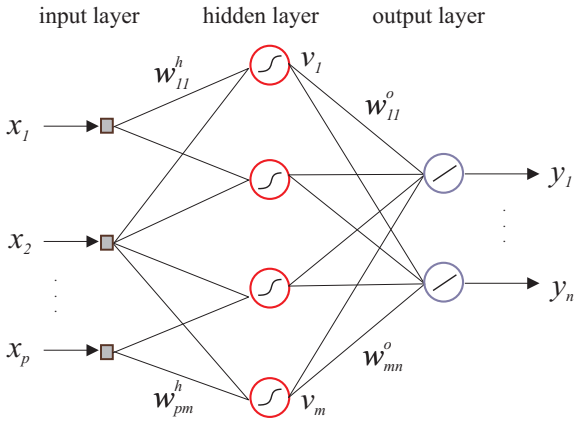


Figure 7.8.: A MNN with one hidden layer, tanh hidden neurons and linear output neurons.

The feedforward computation proceeds in three steps:

1. Compute the activations z_j of the hidden-layer neurons:

$$z_j = \sum_{i=1}^{p} w_{ij}^h x_i + b_j^h, \quad j = 1, 2, \ldots, m\,.$$

   Here, w_{ij}^h and b_j^h are the weight and the bias, respectively.

2. Compute the outputs v_j of the hidden-layer neurons:

$$v_j = \sigma\left(z_j\right), \quad j = 1, 2, \ldots, m\,.$$

3. Compute the outputs y_l of output-layer neurons (and thus of the entire network):

$$y_l = \sum_{j=1}^{h} w_{jl}^o v_j + b_l^o, \quad l = 1, 2, \ldots, n\,.$$

Here, w_{ij}^o and b_j^o are the weight and the bias, respectively.

These three steps can be conveniently written in a compact matrix notation:

$$\begin{aligned} \mathbf{Z} &= \mathbf{X}_b \mathbf{W}^h \\ \mathbf{V} &= \sigma(\mathbf{Z}) \\ \mathbf{Y} &= \mathbf{V}_b \mathbf{W}^o \end{aligned}$$

with $\mathbf{X}_b = [\mathbf{X}\ \mathbf{1}]$ and $\mathbf{V}_b = [\mathbf{V}\ \mathbf{1}]$ and

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix}$$

are the input data and the corresponding output of the net, respectively.

7.6.2. Approximation Properties

Multi-layer neural nets can approximate a large class of functions to a desired degree of accuracy. Loosely speaking, this is because by the superposition of weighted sigmoidal-type functions rather complex mappings can be obtained. As an example, consider a simple MNN with one input, one output and one hidden layer with two *tanh* neurons. The output of this network is given by

$$y = w_1^o \tanh\left(w_1^h x + b_1^h\right) + w_2^o \tanh\left(w_2^h x + b_2^h\right)\,.$$

In Figure 7.9 the three feedforward computation steps are depicted.

Note that two neurons are already able to represent a relatively complex nonmonotonic function. This capability of multi-layer neural nets has formally been stated by Cybenko (1989):

A feedforward neural net with at least one hidden layer with sigmoidal activation functions can approximate any continuous nonlinear function $\mathbb{R}^p \to \mathbb{R}^n$ arbitrarily well on a compact set, provided that sufficient number of hidden neurons are available.

This result is, however, not constructive, which means that is does not say how many neurons are needed, how to determine the weights, etc. Many other function approximators exist, such as polynomials, Fourier series, wavelets, etc. Neural nets, however, are very efficient in terms of the achieved approximation accuracy for given number of neurons. This has been shown by Barron (1993):

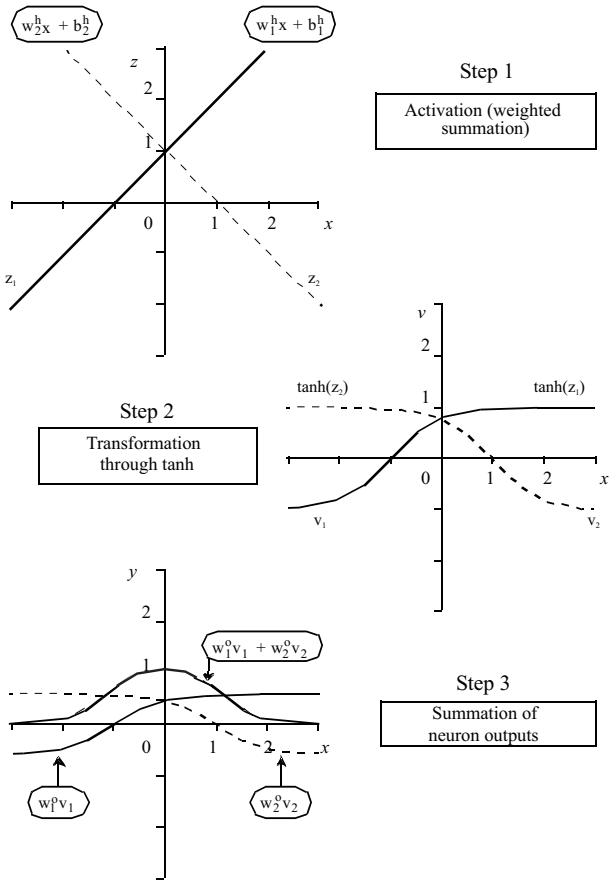


Figure 7.9.: Feedforward computation steps in a multilayer network with two neurons.

A feedforward neural net with one hidden layer with sigmoidal activation functions can achieve an integrated squared error (for smooth functions) of the order

$$J = \mathcal{O}\left(\frac{1}{h}\right)$$

independently of the dimension of the input space p, where h denotes the number of hidden neurons.

For basis function expansion models (polynomials, trigonometric expansion, singleton fuzzy model, etc.) with h terms, in which only the parameters of the linear combination are adjusted, it holds that

$$J = \mathcal{O}\left(\frac{1}{h^{2/p}}\right)$$

where p is the dimension of the input.

Example 7.1 (Approximation Accuracy) To illustrate the difference between the approximation capabilities of sigmoidal nets and basis function expansions (e.g., polynomials), consider two input dimensions p:

i) $p = 2$ (function of two variables):

$$\text{polynomial} \quad J = \mathcal{O}\left(\tfrac{1}{h^{2/2}}\right) = \mathcal{O}\left(\tfrac{1}{h}\right)$$

$$\text{neural net} \quad J = \mathcal{O}\left(\tfrac{1}{h}\right)$$

Hence, for $p = 2$, there is no difference in the accuracy–complexity relationship between sigmoidal nets and basis function expansions.

ii) $p = 10$ (function of ten variables) and $h = 21$:

$$\text{polynomial} \quad J = \mathcal{O}(\tfrac{1}{21^{2/10}}) = 0.54$$

$$\text{neural net} \quad J = \mathcal{O}(\tfrac{1}{21}) = 0.048$$

The approximation accuracy of the sigmoidal net is thus in the order of magnitude better. Let us see, how many terms are needed in a basis function expansion (e.g., the order of a polynomial) in order to achieve the same accuracy as the sigmoidal net:

$$\mathcal{O}(\frac{1}{h_n}) = \mathcal{O}(\frac{1}{h_b})$$

$$h_n = h_b^{1/p} \quad \Rightarrow \quad h_b = \sqrt{h_n^p} = \sqrt{21^{10}} \approx 4 \cdot 10^6$$

□

Multi-layer neural networks are thus, at least in theory, able to approximate other functions. The question is how to determine the appropriate structure (number of hidden layers, number of neurons) and parameters of the network.

A network with one hidden layer is usually sufficient (theoretically always sufficient). More layers can give a better fit, but the training takes longer time. Choosing the right number of neurons in the hidden layer is essential for a good result. Too few neurons give a poor fit, while too many neurons result in overtraining of the net (poor generalization to unseen data). A compromise is usually sought by trial and error methods.

7.6.3. Training, Error Backpropagation

Training is the adaptation of weights in a multi-layer network such that the error between the desired output and the network output is minimized. Assume that a set of N data patterns is available:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} \mathbf{d}_1^T \\ \mathbf{d}_2^T \\ \vdots \\ \mathbf{d}_N^T \end{bmatrix}$$

Here, $\mathbf{x}$ is the input to the net and $\mathbf{d}$ is the desired output. The training proceeds in two steps:

1. *Feedforward computation.* From the network inputs x_i, $i = 1, \ldots, N$, the hidden layer activations, outputs and the network outputs are computed:

$$\begin{aligned} \mathbf{Z} &= \mathbf{X}_b \mathbf{W}^h, \quad \mathbf{X}_b = [\mathbf{X}\ \mathbf{1}] \\ \mathbf{V} &= \sigma(\mathbf{Z}) \\ \mathbf{Y} &= \mathbf{V}_b \mathbf{W}^o, \quad \mathbf{V}_b = [\mathbf{V}\ \mathbf{1}] \end{aligned}$$

2. *Weight adaptation.* The output of the network is compared to the desired output. The difference of these two values called the error:

$$\mathbf{E} = \mathbf{D} - \mathbf{Y}$$

   This error is used to adjust the weights in the net via the minimization of the following cost function (sum of squared errors):

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2} \sum_{k=1}^{N} \sum_{j=1}^{l} e_{kj}^2 = \text{trace}\,(\mathbf{E}\mathbf{E}^T) \\ \mathbf{w} &= \left[\mathbf{W}^h\ \mathbf{W}^o\right] \end{aligned}$$

The training of a MNN is thus formulated as a *nonlinear optimization problem* with respect to the weights. Various methods can be applied:

- Error backpropagation (first-order gradient).

- Newton, Levenberg-Marquardt methods (second-order gradient).
- Conjugate gradients.
- Variable projection.
- Genetic algorithms, many others ...

First-order gradient methods use the following update rule for the weights:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \alpha(n)\nabla J(\mathbf{w}(n)), \tag{7.2}$$

where $\mathbf{w}(n)$ is the vector with weights at iteration n, $\alpha(n)$ is a (variable) learning rate and $\nabla J(\mathbf{w})$ is the Jacobian of the network

$$\nabla J(\mathbf{w}) = \left[\frac{\partial J(\mathbf{w})}{\partial w_1}, \frac{\partial J(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_M}\right]^T.$$

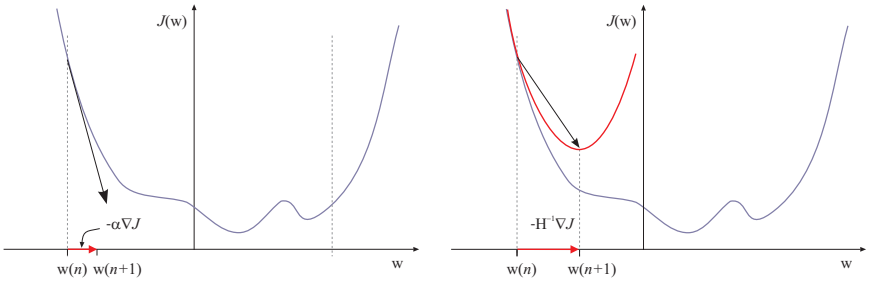
The nonlinear optimization problem is thus solved by using the first term of its Taylor series expansion (the gradient). Second-order gradient methods make use of the second term (the curvature) as well:

$$J(\mathbf{w}) \approx J(\mathbf{w}_0) + \nabla J(\mathbf{w}_0)^T(\mathbf{w} - \mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T\mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0),$$

where $\mathbf{H}(\mathbf{w}_0)$ is the Hessian in a given point $\mathbf{w}_0$ in the weight space. After a few steps of derivations, the update rule for the weights appears to be:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mathbf{H}^{-1}(\mathbf{w}(n))\nabla J(\mathbf{w}(n)) \tag{7.3}$$

The difference between (7.2) and (7.3) is basically the size of the gradient-descent step. This is schematically depicted in Figure 7.10.



(a) first-order gradient (b) second-order gradient

Figure 7.10.: First-order and second-order gradient-descent optimization.

Second order methods are usually more effective than first-order ones. Here, we will, however, present the *error backpropagation*, (first-order gradient method) which is easier to grasp and then the step toward understanding second-order methods is small.

The main idea of backpropagation (BP) can be expressed as follows:

- compute errors at the outputs,
- adjust output weights,
- propagate error backwards through the net and adjust hidden-layer weights.

We will derive the BP method for processing the data set pattern by pattern, which is suitable for both on-line and off-line training. First, consider the output-layer weights and then the hidden-layer ones.

Output-Layer Weights

Consider a neuron in the output layer as depicted in Figure 7.11.

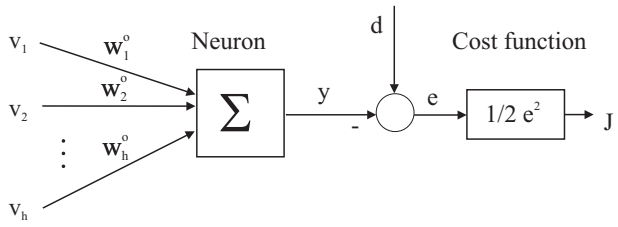


Figure 7.11.: Output-layer neuron.

The cost function is given by:

$$J = \frac{1}{2}\sum_{l} e_l^2, \quad \text{with } e_l = d_l - y_l, \quad \text{and } y_l = \sum_j w_j^o v_j \tag{7.4}$$

The Jacobian is:

$$\frac{\partial J}{\partial w_{jl}^o} = \frac{\partial J}{\partial e_l} \cdot \frac{\partial e_l}{\partial y_l} \cdot \frac{\partial y_l}{\partial w_{jl}^o} \tag{7.5}$$

with the partial derivatives:

$$\frac{\partial J}{\partial e_l} = e_l, \quad \frac{\partial e_l}{\partial y_l} = -1, \quad \frac{\partial y_l}{\partial w_{jl}^o} = v_j \tag{7.6}$$

Hence, for the output layer, the Jacobian is:

$$\frac{\partial J}{\partial w_{jl}^o} \;=\; -v_j e_l \,.$$

From (7.2), the update law for the output weights follows:

$$w_{jl}^o(n+1) = w_{jl}^o(n) + \alpha(n) v_j e_l \,. \tag{7.7}$$

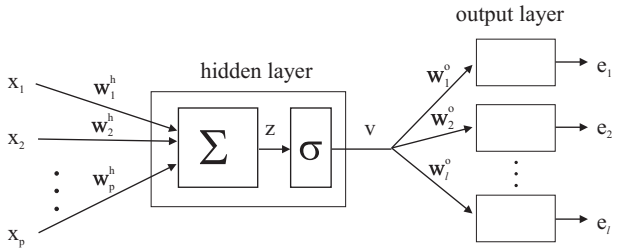


Figure 7.12.: Hidden-layer neuron.

Hidden-Layer Weights

Consider a neuron in the hidden layer as depicted in Figure 7.12.

The Jacobian is:

$$\frac{\partial J}{\partial w_{ij}^h} = \frac{\partial J}{\partial v_j} \cdot \frac{\partial v_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}^h} \tag{7.8}$$

with the partial derivatives (after some computations):

$$\frac{\partial J}{\partial v_j} = \sum_l -e_l w_{jl}^o, \quad \frac{\partial v_j}{\partial z_j} = \sigma_j'(z_j), \quad \frac{\partial z_j}{\partial w_{ij}^h} = x_i \tag{7.9}$$

The derivation of the above expression is quite straightforward and we leave it as an exercise for the reader. Substituting into (7.8) gives the Jacobian

$$\frac{\partial J}{\partial w_{ij}^h} = -x_i \cdot \sigma_j'(z_j) \cdot \sum_l e_l w_{jl}^o$$

From (7.2), the update law for the hidden weights follows:

$$w_{ij}^h(n+1) = w_{ij}^h(n) + \alpha(n) x_i \cdot \sigma_j'(z_j) \cdot \sum_l e_l w_{jl}^o \,. \tag{7.10}$$

From this equation, one can see that the error is propagated from the output layer to the hidden layer, which gave rise to the name "backpropagation". The algorithm is summarized in Algorithm 7.1.

Algorithm 7.1 backpropagation

Initialize the weights (at random).

Step 1: Present the inputs and desired outputs.

Step 2: Calculate the actual outputs and errors.

Step 3: Compute gradients and update the weights:

$$\begin{aligned} w_{jl}^o &:= w_{jl}^o + \alpha v_j e_l \\ w_{ij}^h &:= w_{ij}^h + \alpha x_i \cdot \sigma_j'(z_j) \cdot \sum_l e_l w_{jl}^o \end{aligned}$$

Repeat by going to *Step 1*.

In this approach, the data points are presented for learning one after another. This is mainly suitable for on-line learning. However, it still can be applied if a whole batch of data is available for off-line learning.

The presentation of the whole data set is called an *epoch*. Usually, several learning epochs must be applied in order to achieve a good fit. From a computational point of view, it is more effective to present the data set as the whole batch. The backpropagation learning formulas are then applied to vectors of data instead of the individual samples.

7.7. Radial Basis Function Network

The radial basis function network (RBFN) is a two layer network. There are two main differences from the multi-layer sigmoidal network:

- The activation function in the hidden layer is a radial basis function rather than a sigmoidal function. Radial basis functions are explained below.
- Adjustable weights are only present in the output layer. The connections from the input layer to the hidden layer are fixed (unit weights). Instead, the parameters of the radial basis functions are tuned.

The output layer neurons are linear. The RBFN thus belongs to models of the basis function expansion type, similar to the singleton model of Section 3.3. It realizes the following function $f \to \mathbb{R}^p \to \mathbb{R}$:

$$y = f(\mathbf{x}) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{x}, \mathbf{c}_i) \tag{7.11}$$

Some common choices for the basis functions $\phi_i(\mathbf{x}, \mathbf{c}_i) = \phi_i(\|\mathbf{x} - \mathbf{c}_i\|) = \phi_i(r)$ are:

- $\phi(r) = \exp(-r^2/\rho^2)$, a Gaussian function
- $\phi(r) = r^2 \log(r)$, a thin-plate-spline function
- $\phi(r) = r^2$, a quadratic function
- $\phi(r) = (r^2 + \rho^2)^{\frac{1}{2}}$, a multiquadratic function

Figure 7.13 depicts the architecture of a RBFN.

The free parameters of RBFNs are the output weights w_i and the parameters of the basis functions (centers $\mathbf{c}_i$ and radii ρ_i).

The network's output (7.11) is linear in the weights w_i, which thus can be estimated by least-squares methods. For each data point $\mathbf{x}_k$, first the outputs of the neurons are computed:

$$v_{ki} = \phi_i(\mathbf{x}, \mathbf{c}_i)\,.$$

As the output is linear in the weights w_i, we can write the following matrix equation for the whole data set:

$$\mathbf{d} = \mathbf{V}\mathbf{w},$$

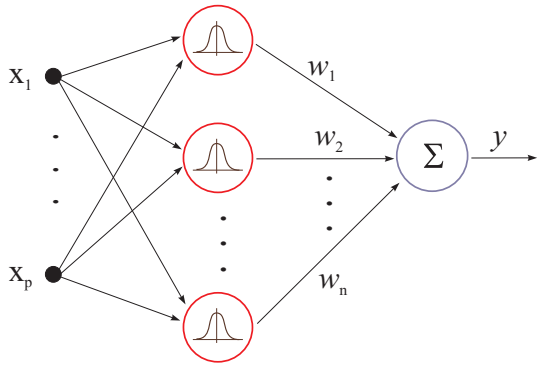


Figure 7.13.: Radial basis function network.

where $\mathbf{V} = [v_{ki}]$ is a matrix of the neurons' outputs for each data point and $\mathbf{d}$ is a vector of the desired RBFN outputs. The least-square estimate of the weights $\mathbf{w}$ is:

$$\mathbf{w} = \left[\mathbf{V}^T\mathbf{V}\right]^{-1}\mathbf{V}^T\mathbf{y}$$

The training of the RBF parameters $\mathbf{c}_i$ and ρ_i leads to a nonlinear optimization problem that can be solved, for instance, by the methods given in Section 7.6.3. Initial center positions are often determined by clustering (see Chapter 4).

7.8. Summary and Concluding Remarks

Artificial neural nets, originally inspired by the functionality of biological neural networks can learn complex functional relations by generalizing from a limited amount of training data. Neural nets can thus be used as (black-box) models of nonlinear, multivariable static and dynamic systems and can be trained by using input–output data observed on the system. Many different architectures have been proposed. In systems modeling and control, most commonly used are the multi-layer feedforward neural network and the radial basis function network. Effective training algorithms have been developed for these networks.

7.9. Problems

1. What has been the original motivation behind artificial neural networks? Give at least two examples of control engineering applications of artificial neural networks.

2. Draw a block diagram and give the formulas for an artificial neuron. Explain all terms and symbols.

3. Give at least three examples of activation functions.

4. Explain the term "training" of a neural network.

5. What are the steps of the backpropagation algorithm? With what neural network architecture is this algorithm used?

6. Explain the difference between first-order and second-order gradient optimization.

7. Derive the backpropagation rule for an output neuron with a sigmoidal activation function.

8. What are the differences between a multi-layer feedforward neural network and the radial basis function network?

9. Consider a dynamic system $y(k+1) = f\big(y(k), y(k-1), u(k), u(k-1)\big)$, where the function f is unknown. Suppose, we wish to approximate f by a neural network trained by using a sequence of N input–output data samples measured on the unknown system, $\{(u(k), y(k)) | k = 0, 1, \ldots, N\}$.
   a) Choose a neural network architecture, draw a scheme of the network and define its inputs and outputs.
   b) What are the free parameters that must be trained (optimized) such that the network fits the data well?
   c) Define a cost function for the training (by a formula) and name examples of two methods you could use to train the network's parameters.

8. Control Based on Fuzzy and Neural Models

This chapter addresses the design of a nonlinear controller based on an available fuzzy or neural model of the process to be controlled. Some presented techniques are generally applicable to both fuzzy and neural models (predictive control, feedback linearization), others are based on specific features of fuzzy models (gain scheduling, analytic inverse).

8.1. Inverse Control

The simplest approach to model-based design a controller for a nonlinear process is *inverse control*. It can be applied to a class of systems that are open-loop stable (or that are stabilizable by feedback) and whose inverse is stable as well, i.e., the system does not exhibit nonminimum phase behavior.

For simplicity, the approach is here explained for SISO models without transport delay from the input to the output. The available neural or fuzzy model can be written as a general nonlinear model:

$$y(k+1) = f\big(\mathbf{x}(k), u(k)\big)\,. \tag{8.1}$$

The inputs of the model are the current state $\mathbf{x}(k) = [y(k), \ldots, y(k-n_y+1), u(k-1), \ldots, u(k-n_u+1)]^T$ and the current input $u(k)$. The model predicts the system's output at the next sample time, $y(k+1)$. The function f represents the nonlinear mapping of the fuzzy or neural model.

The objective of inverse control is to compute for the current state $\mathbf{x}(k)$ the control input $u(k)$, such that the system's output at the next sampling instant is equal to the desired (reference) output $r(k+1)$. This can be achieved if the process model (8.1) can be inverted according to:

$$u(k) = f^{-1}(\mathbf{x}(k), r(k+1))\,. \tag{8.2}$$

Here, the reference $r(k+1)$ was substituted for $y(k+1)$. The inverse model can be used as an open-loop feedforward controller, or as an open-loop controller with feedback of the process' output (called open-loop feedback controller). The difference between the two schemes is the way in which the state $\mathbf{x}(k)$ is updated.

8.1.1. Open-Loop Feedforward Control

The state $\mathbf{x}(k)$ of the inverse model (8.2) is updated using the output of the model (8.1), see Figure 8.1. As no feedback from the process output is used, stable control is guaranteed

for open-loop stable, minimum-phase systems. However, a model-plant mismatch or a disturbance d will cause a steady-state error at the process output. This error can be compensated by some kind of feedback, using, for instance, the IMC scheme presented in Section 8.1.5.

Besides the model and the controller, the control scheme contains a reference-shaping filter. This is usually a first-order or a second-order reference model, whose task is to generate the desired dynamics and to avoid peaks in the control action for step-like references.

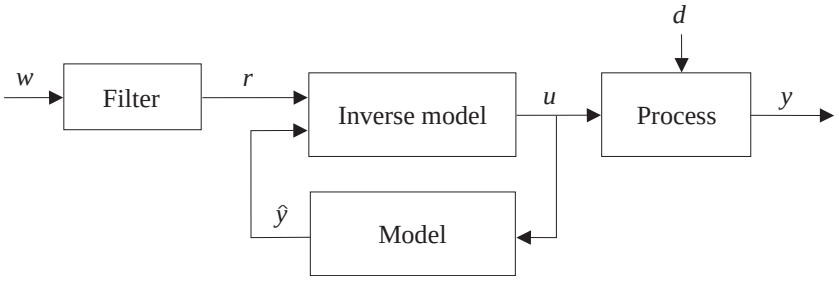


Figure 8.1.: Open-loop feedforward inverse control.

8.1.2. Open-Loop Feedback Control

The input $\mathbf{x}(k)$ of the inverse model (8.2) is updated using the output of the process itself, see Figure 8.2. The controller, in fact, operates in an open loop (does not use the error between the reference and the process output), but the current output $y(k)$ of the process is used at each sample to update the internal state $\mathbf{x}(k)$ of the controller. This can improve the prediction accuracy and eliminate offsets. At the same time, however, the direct updating of the model state may not be desirable in the presence of noise or a significant model–plant mismatch, in which cases it can cause oscillations or instability. Also this control scheme contains the reference-shaping filter.

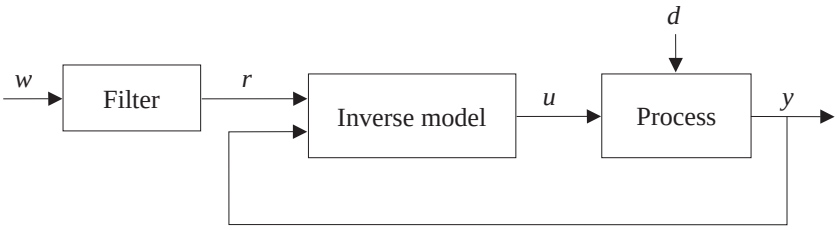


Figure 8.2.: Open-loop feedback inverse control.

8.1.3. Computing the Inverse

Generally, it is difficult to find the inverse function f^{-1} in an analytical from. It can, however, always be found by numerical optimization (search). Define an objective function:

$$J(u(k)) = (r(k+1) - f(\mathbf{x}(k), u(k)))^2 \ . \tag{8.3}$$

The minimization of J with respect to $u(k)$ gives the control corresponding to the inverse function (8.2), if it exists, or the best approximation of it otherwise. A wide variety of optimization techniques can be applied (such as Newton or Levenberg-Marquardt). This approach directly extends to MIMO systems. Its main drawback, however, is the computational complexity due to the numerical optimization that must be carried out on-line.

Some special forms of (8.1) can be inverted analytically. Examples are an input-affine Takagi–Sugeno (TS) model and a singleton model with triangular membership functions fro $u(k)$.

Affine TS Model

Consider the following input–output Takagi–Sugeno (TS) fuzzy model:

$$\begin{aligned}\mathcal{R}_i: \quad & \textbf{If } y(k) \text{ is } A_{i1} \textbf{ and } \ldots \textbf{ and } y(k-n_y+1) \text{ is } A_{in_y} \textbf{ and} \\ & u(k-1) \text{ is } B_{i2} \textbf{ and } \ldots \textbf{ and } u(k-n_u+1) \text{ is } B_{in_u} \textbf{ then} \\ & y_i(k+1) = \sum_{j=1}^{n_y} a_{ij} y(k-j+1) + \sum_{j=1}^{n_u} b_{ij} u(k-j+1) + c_i, \end{aligned} \tag{8.4}$$

where $i = 1, \ldots, K$ are the rules, A_{il}, B_{il} are fuzzy sets, and a_{ij}, b_{ij}, c_i are crisp consequent (then-part) parameters. Denote the antecedent variables, i.e., the lagged outputs and inputs (excluding $u(k)$), by:

$$\mathbf{x}(k) = [y(k), y(k-1), \ldots, y(k-n_y+1), u(k-1), \ldots, u(k-n_u+1)]\,. \tag{8.5}$$

The output $y(k+1)$ of the model is computed by the weighted mean formula:

$$y(k+1) = \frac{\sum_{i=1}^{K} \beta_i(\mathbf{x}(k)) y_i(k+1)}{\sum_{i=1}^{K} \beta_i(\mathbf{x}(k))}, \tag{8.6}$$

where β_i is the degree of fulfillment of the antecedent given by:

$$\begin{aligned}\beta_i(\mathbf{x}(k)) \;=\; & \mu_{A_{i1}}(y(k)) \wedge \ldots \wedge \mu_{A_{in_y}}(y(k-n_y+1)) \wedge \\ & \mu_{B_{i2}}(u(k-1)) \wedge \ldots \wedge \mu_{B_{in_u}}(u(k-n_u+1))\,. \end{aligned} \tag{8.7}$$

As the antecedent of (8.4) does not include the input term $u(k)$, the model output $y(k+1)$ is affine in the input $u(k)$. To see this, denote the normalized degree of fulfillment

$$\lambda_i(\mathbf{x}(k)) = \frac{\beta_i(\mathbf{x}(k))}{\sum_{j=1}^{K} \beta_j(\mathbf{x}(k))}, \tag{8.8}$$

and substitute the consequent of (8.4) and the λ_i of (8.8) into (8.6):

$$\begin{aligned} y(k+1) \;=\; & \sum_{i=1}^{K} \lambda_i(\mathbf{x}(k)) \left[\sum_{j=1}^{n_y} a_{ij} y(k-j+1) + \sum_{j=2}^{n_u} b_{ij} u(k-j+1) + c_i \right] + \\ & + \sum_{i=1}^{K} \lambda_i(\mathbf{x}(k)) b_{i1} u(k) \end{aligned} \tag{8.9}$$

This is a nonlinear input-affine system which can in general terms be written as:

$$y(k+1) = g\left(\mathbf{x}(k)\right) + h(\mathbf{x}(k))u(k)\,. \tag{8.10}$$

Given the goal that the model output at time step $k+1$ should equal the reference output, $y(k+1) = r(k+1)$, the corresponding input, $u(k)$, is computed by a simple algebraic manipulation:

$$u(k) = \frac{r(k+1) - g\left(\mathbf{x}(k)\right)}{h(\mathbf{x}(k))}\,. \tag{8.11}$$

In terms of eq. (8.9) we obtain the eventual inverse-model control law:

$$u(k) = \frac{r(k+1) - \sum_{i=1}^{K} \lambda_i(\mathbf{x}(k)) \left[\sum_{j=1}^{n_y} a_{ij} y(k-j+1) + \sum_{j=2}^{n_u} b_{ij} u(k-j+1) + c_i\right]}{\sum_{i=1}^{K} \lambda_i(\mathbf{x}(k)) b_{i1}}\,. \tag{8.12}$$

Singleton Model

Consider a SISO singleton fuzzy model. In this section, the rule index is omitted, in order to simplify the notation. The considered fuzzy rule is then given by the following expression:

$$\begin{aligned} &\textbf{If } y(k) \text{ is } A_1 \textbf{ and } y(k-1) \text{ is } A_2 \textbf{ and } \ldots \textbf{ and } y(k-n_y+1) \text{ is } A_{n_y} \\ &\textbf{and } u(k) \text{ is } B_1 \textbf{ and } \ldots \textbf{ and } u(k-n_u+1) \text{ is } B_{n_u} \\ &\textbf{then } y(k+1) \text{ is } c, \end{aligned} \tag{8.13}$$

where $A_1, \ldots, A_{n_y}$ and $B_1, \ldots, B_{n_u}$ are fuzzy sets and c is a singleton, see (3.37). Use the state vector $\mathbf{x}(k)$ introduced in (8.5), containing the $n_u - 1$ past inputs, the $n_y - 1$ past outputs and the current output, i.e., all the antecedent variables in (8.13). The corresponding fuzzy sets are composed into one multidimensional state fuzzy set X, by applying a t-norm operator on the Cartesian product space of the state variables: $X = A_1 \times \cdots \times A_{n_y} \times B_2 \times \cdots \times B_{n_u}$. To simplify the notation, substitute B for B_1. Rule (8.13) now can be written by:

$$\textbf{If } \mathbf{x}(k) \text{ is } X \textbf{ and } u(k) \text{ is } B \textbf{ then } y(k+1) \text{ is } c\,. \tag{8.14}$$

Note that the transformation of (8.13) into (8.14) is only a formal simplification of the rule base which does not change the order of the model dynamics, since $\mathbf{x}(k)$ is a vector and X is a multidimensional fuzzy set. Let M denote the number of fuzzy sets X_i defined for the state $\mathbf{x}(k)$ and N the number of fuzzy sets B_j defined for the input $u(k)$. Assume that the rule base consists of all possible combinations of sets X_i and B_j, the total number of rules is then $K = MN$. The entire rule base can be represented as a table:

	$u(k)$			
$\mathbf{x}(k)$	B_1	B_2	$\ldots$	B_N
X_1	c_{11}	c_{12}	$\ldots$	c_{1N}
X_2	c_{21}	c_{22}	$\ldots$	c_{2N}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
X_M	c_{M1}	c_{M2}	$\ldots$	c_{MN}

(8.15)

By using the product t-norm operator, the degree of fulfillment of the rule antecedent $\beta_{ij}(k)$ is computed by:

$$\beta_{ij}(k) \;\; = \mu_{X_i}(\mathbf{x}(k)) \cdot \mu_{B_j}(u(k)) \tag{8.16}$$

The output $y(k+1)$ of the model is computed as an average of the consequents c_{ij} weighted by the normalized degrees of fulfillment β_{ij}:

$$\begin{aligned} y(k+1) &= \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}\beta_{ij}(k)\cdot c_{ij}}{\sum_{i=1}^{M}\sum_{j=1}^{N}\beta_{ij}(k)} = \\ &= \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}\mu_{X_i}(\mathbf{x}(k))\cdot\mu_{B_j}(u(k))\cdot c_{ij}}{\sum_{i=1}^{M}\sum_{j=1}^{N}\mu_{X_i}(\mathbf{x}(k))\cdot\mu_{B_j}(u(k))}\,. \end{aligned} \tag{8.17}$$

Example 8.1 Consider a fuzzy model of the form $y(k+1) = f(y(k), y(k-1), u(k))$ where two linguistic terms $\{$ *low*, *high*$\}$ are used for $y(k)$ and $y(k-1)$ and three terms $\{$ *small*, *medium*, *large*$\}$ for $u(k)$. The complete rule base consists of $2 \times 2 \times 3 = 12$ rules:

If $y(k)$ is *low* and $y(k-1)$ is *low* and $u(k)$ is *small* **then** $y(k+1)$ is c_{11}
If $y(k)$ is *low* and $y(k-1)$ is *low* and $u(k)$ is *medium* **then** $y(k+1)$ is c_{12}
...
If $y(k)$ is *high* and $y(k-1)$ is *high* and $u(k)$ is *large* **then** $y(k+1)$ is c_{43}

In this example $\mathbf{x}(k) = [y(k), y(k-1)]$, $X_i \in \{(low \times low), (low \times high), (high \times low), (high \times high)\}$, $M = 4$ and $N = 3$. The rule base is represented by the following table:

	$u(k)$		
$\mathbf{x}(k)$	*small*	*medium*	*large*
X_1 (*low* × *low*)	c_{11}	c_{12}	c_{13}
X_2 (*low* × *high*)	c_{21}	c_{22}	c_{23}
X_3 (*high* × *low*)	c_{31}	c_{32}	c_{33}
X_4 (*high* × *high*)	c_{41}	c_{42}	c_{43}

(8.18)

□

The inversion method requires that the antecedent membership functions $\mu_{B_j}(u(k))$ are triangular and form a partition, i.e., fulfill:

$$\sum_{j=1}^{N}\mu_{B_j}(u(k)) = 1\,. \tag{8.19}$$

The basic idea is the following. For each particular state $\mathbf{x}(k)$, the multivariate mapping (8.1) is reduced to a univariate mapping

$$y(k+1) = f_x(u(k)), \tag{8.20}$$

where the subscript x denotes that f_x is obtained for the particular state $\mathbf{x}$. From this mapping, which is piecewise linear, the inverse mapping $u(k) = f_x^{-1}(r(k+1))$ can be easily found, provided the model is invertible. This invertibility is easy to check for univariate functions. First, using (8.19), the output equation of the model (8.17) simplifies to:

$$\begin{aligned} y(k+1) &= \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}\mu_{X_i}(\mathbf{x}(k))\cdot\mu_{B_j}(u(k))\cdot c_{ij}}{\sum_{i=1}^{M}\sum_{j=1}^{N}\mu_{X_i}(\mathbf{x}(k))\mu_{B_j}(u(k))} \\ &= \sum_{i=1}^{M}\sum_{j=1}^{N}\lambda_i(\mathbf{x}(k))\cdot\mu_{B_j}(u(k))\cdot c_{ij} \\ &= \sum_{j=1}^{N}\mu_{B_j}(u(k))\sum_{i=1}^{M}\lambda_i(\mathbf{x}(k))\cdot c_{ij}\,. \end{aligned} \tag{8.21}$$

where $\lambda_i(\mathbf{x}(k))$ is the normalized degree of fulfillment of the state part of the antecedent:

$$\lambda_i(\mathbf{x}(k)) = \frac{\mu_{X_i}(\mathbf{x}(k))}{\sum_{j=1}^{K}\mu_{X_j}(\mathbf{x}(k))}\,. \tag{8.22}$$

As the state $\mathbf{x}(k)$ is available, the latter summation in (8.21) can be evaluated, yielding:

$$y(k+1) = \sum_{j=1}^{N}\mu_{B_j}(u(k))c_j, \tag{8.23}$$

where

$$c_j = \sum_{i=1}^{M}\lambda_i(\mathbf{x}(k))\cdot c_{ij}\,. \tag{8.24}$$

This is an equation of a singleton model with input $u(k)$ and output $y(k+1)$:

$$\textbf{If } u(k) \text{ is } B_j \quad \textbf{then } y(k+1) \text{ is } c_j(k), \qquad j = 1,\ldots,N\,. \tag{8.25}$$

Each of the above rules is inverted by exchanging the antecedent and the consequent, which yields the following rules:

$$\textbf{If } r(k+1) \text{ is } c_j(k) \quad \textbf{then } u(k) \text{ is } B_j \qquad j = 1,\ldots,N\,. \tag{8.26}$$

Here, the reference $r(k+1)$ was substituted for $y(k+1)$. Since $c_j(k)$ are singletons, it is necessary to interpolate between the consequents $c_j(k)$ in order to obtain $u(k)$. This interpolation is accomplished by fuzzy sets C_j with triangular membership functions:

$$\mu_{C_1}(r) = \max\bigl(0, \min(1, \frac{c_2 - r}{c_2 - c_1})\bigr), \tag{8.27a}$$

$$\mu_{C_j}(r) = \max\bigl(0, \min(\frac{r - c_{j-1}}{c_j - c_{j-1}}, \frac{c_{j+1} - r}{c_{j+1} - c_j})\bigr), \quad 1 < j < N, \tag{8.27b}$$

$$\mu_{C_N}(r) = \max\bigl(0, \min(\frac{r - c_{N-1}}{c_N - c_{N-1}}, 1)\bigr)\,. \tag{8.27c}$$

The output of the inverse controller is thus given by:

$$u(k) = \sum_{j=1}^{N} \mu_{C_j}\big(r(k+1)\big) b_j, \tag{8.28}$$

where b_j are the cores of B_j. The inversion is thus given by equations (8.22), (8.27) and (8.28). It can be verified that the series connection of the controller and the inverse model, shown in Figure 8.3, gives an identity mapping (perfect control)

$$y(k+1) = f_x(u(k)) = f_x\Big(f_x^{-1}\big(r(k+1)\big)\Big) = r(k+1), \tag{8.29}$$

when $u(k)$ exists such that $r(k+1) = f\big(\mathbf{x}(k), u(k)\big)$. When no such $u(k)$ exists, the difference

$$\Big| r(k+1) - f_x\Big(f_x^{-1}\big(r(k+1)\big)\Big) \Big|$$

is the least possible. The proof is left as an exercise.

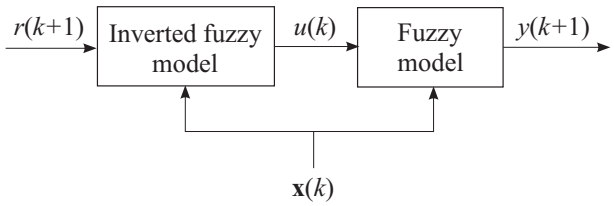


Figure 8.3.: Series connection of the fuzzy model and the controller based on the inverse of this model.

Apart from the computation of the membership degrees, both the model and the controller can be implemented using standard matrix operations and linear interpolations, which makes the algorithm suitable for real-time implementation.

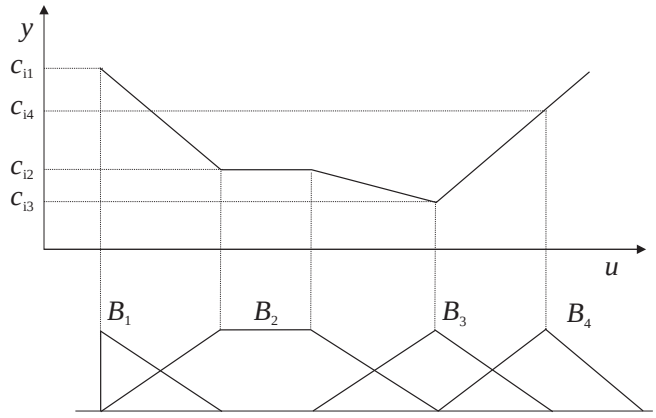


Figure 8.4.: Example of a noninvertible singleton model.

For a noninvertible rule base (see Figure 8.4), a set of possible control commands can be found by splitting the rule base into two or more invertible parts. For each part, a control

action is found by inversion. Among these control actions, only one has to be selected, which requires some additional criteria, such as minimal control effort (minimal $u(k)$ or $|u(k) - u(k-1)|$, for instance).

The invertibility of the fuzzy model can be checked in run-time, by checking the monotonicity of the aggregated consequents c_j with respect to the cores of the input fuzzy sets b_j, see eq. (8.24). This is useful, since nonlinear models can be noninvertible only locally, resulting in a kind of exception in the inversion algorithm. Moreover, for models adapted on line, this check is necessary.

Example 8.2 Consider the fuzzy model from Example 8.1, which is, for convenience, repeated below:

	$u(k)$		
$\mathbf{x}(k)$	*small*	*medium*	*large*
$X_1(low \times low)$	c_{11}	c_{12}	c_{13}
$X_2(low \times high)$	c_{21}	c_{22}	c_{23}
$X_3(high \times low)$	c_{31}	c_{32}	c_{33}
$X_4(high \times high)$	c_{41}	c_{42}	c_{43}

Given the state $\mathbf{x}(k) = [y(k), y(k-1)]$, the degree of fulfillment of the first antecedent proposition "$\mathbf{x}(k)$ is X_i", is calculated as $\mu_{X_i}(\mathbf{x}(k))$. For X_2, for instance, $\mu_{X_2}(\mathbf{x}(k)) = \mu_{\text{low}}(y(k)) \cdot \mu_{\text{high}}(y(k-1))$. Using (8.24), one obtains the cores $c_j(k)$:

$$c_j(k) = \sum_{i=1}^{4} \mu_{X_i}(\mathbf{x}(k)) c_{ij}, \quad j = 1, 2, 3\,. \tag{8.30}$$

An example of membership functions for fuzzy sets C_j, obtained by eq. (8.27), is shown in Figure 8.5:

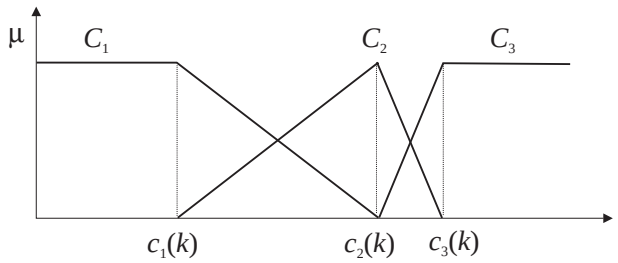


Figure 8.5.: Fuzzy partition created from $c_1(k)$, $c_2(k)$ and $c_3(k)$.

Assuming that $b_1 < b_2 < b_3$, the model is (locally) invertible if $c_1 < c_2 < c_3$ or if $c_1 > c_2 > c_3$. In such a case, the following rules are obtained:

1) **If** $r(k+1)$ is $C_1(k)$ **then** $u(k)$ is B_1
2) **If** $r(k+1)$ is $C_2(k)$ **then** $u(k)$ is B_2
3) **If** $r(k+1)$ is $C_3(k)$ **then** $u(k)$ is B_3

Otherwise, if the model is not invertible, e.g., $c_1 > c_2 < c_3$, the above rule base must be split into two rule bases. The first one contains rules 1 and 2, and the second one rules 2 and 3.

□

8.1.4. Inverting Models with Transport Delays

For models with input delays $y(k+1) = f\big(\mathbf{x}(k), u(k-n_d)\big)$, the inversion cannot be applied directly, as it would give a control action $u(k)$, n_d steps delayed. In order to generate the appropriate value $u(k)$, the model must be inverted $n_d - 1$ samples ahead, i.e., $u(k) = f^{-1}(r(k+n_d+1), \mathbf{x}(k+n_d))$, where

$$\begin{aligned}\mathbf{x}(k+n_d) &= [y(k+n_d), \ldots, y(k+1), \ldots \\ &\quad y(k-n_y+n_d+1), u(k-1), \ldots, u(k-n_u+1)]^T .\end{aligned}$$

The unknown values, $y(k+1), \ldots, y(k+n_d)$, are predicted recursively using the model:

$$\begin{aligned}y(k+i) &= f\big(\mathbf{x}(k+i-1), u(k-n_d+i-1)\big), \\ \mathbf{x}(k+i) &= [y(k+i), \ldots, y(k-n_y+i+1), u(k-n_d+i-1), \ldots \\ &\quad u(k-n_u-n_d+i+1)]^T\end{aligned}$$

for $i = 1, \ldots, n_d$.

8.1.5. Internal Model Control

Disturbances acting on the process, measurement noise and model-plant mismatch cause differences in the behavior of the process and of the model. In open-loop control, this results in an error between the reference and the process output. The *internal model control* scheme (Economou et al., 1986) is one way of compensating for this error.

Figure 8.6 depicts the IMC scheme, which consists of three parts: the controller based on an inverse of the process model, the model itself, and a feedback filter. The control system (dashed box) has two inputs, the reference and the measurement of the process output, and one output, the control action.

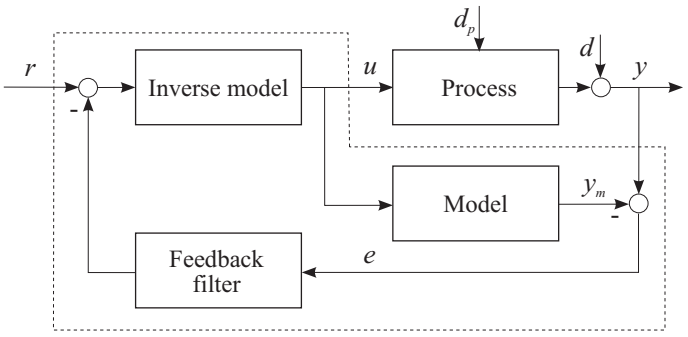


Figure 8.6.: Internal model control scheme.

The purpose of the process model working in parallel with the process is to subtract the effect of the control action from the process output. If the predicted and the measured process outputs are equal, the error e is zero and the controller works in an open-loop configuration. If a disturbance d acts on the process output, the feedback signal e is equal to the influence of the disturbance and is not affected by the effects of the control action. This signal is subtracted from the reference. With a perfect process model, the IMC scheme is hence able to cancel the effect of unmeasured output-additive disturbances.

The feedback filter is introduced in order to filter out the measurement noise and to stabilize the loop by reducing the loop gain for higher frequencies. With nonlinear systems and models, the filter must be designed empirically.

8.2. Model-Based Predictive Control

Model-based predictive control (MBPC) is a general methodology for solving control problems in the time domain. It is based on three main concepts:

1. A model is used to predict the process output at future discrete time instants, over a *prediction horizon.*
2. A sequence of future control actions is computed over a *control horizon* by minimizing a given objective function.
3. Only the first control action in the sequence is applied, the horizons are moved towards the future and optimization is repeated. This is called the *receding horizon* principle.

Because of the optimization approach and the explicit use of the process model, MBPC can realize multivariable optimal control, deal with nonlinear processes, and can efficiently handle constraints.

8.2.1. Prediction and Control Horizons

The future process outputs are predicted over the *prediction horizon* H_p using a model of the process. The predicted output values, denoted $\hat{y}(k+i)$ for $i = 1, \ldots, H_p$, depend on the state of the process at the current time k and on the future control signals $u(k+i)$ for $i = 0, \ldots, H_c - 1$, where $H_c \leq H_p$ is the *control horizon.* The control signal is manipulated only within the control horizon and remains constant afterwards, i.e., $u(k+i) = u(k+H_c-1)$ for $i = H_c, \ldots, H_p - 1$, see Figure 8.7.

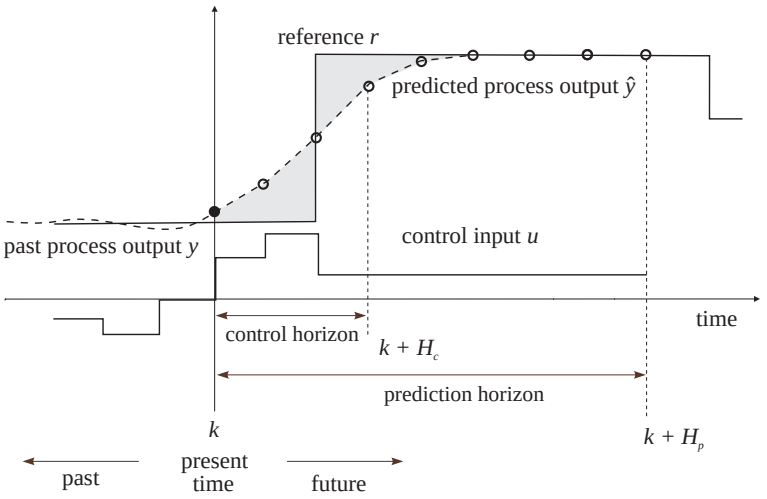


Figure 8.7.: The basic principle of model-based predictive control.

8.2.2. Objective Function

The sequence of future control signals $\mathbf{u}(k+i)$ for $i = 0, 1, \ldots, H_c - 1$ is usually computed by optimizing the following quadratic cost function (Clarke et al., 1987):

$$J = \sum_{i=1}^{H_p} \|(\mathbf{r}(k+i) - \hat{\mathbf{y}}(k+i))\|^2_{\mathbf{P}_i} + \sum_{i=1}^{H_c} \|(\Delta\mathbf{u}(k+i-1))\|^2_{\mathbf{Q}_i} . \tag{8.31}$$

The first term accounts for minimizing the variance of the process output from the reference, while the second term represents a penalty on the control effort (related, for instance, to energy). The latter term can also be expressed by using u itself. $\mathbf{P}_i$ and $\mathbf{Q}_i$ are positive definite weighting matrices that specify the importance of two terms in (8.31) relative to each other and to the prediction step. Additional terms can be included in the cost function to account for other control criteria.

For systems with a dead time of n_d samples, only outputs from time $k+n_d$ are considered in the objective function, because outputs before this time cannot be influenced by the control signal $u(k)$. Similar reasoning holds for nonminimum phase systems.

"Hard" constraints, e.g., level and rate constraints of the control input, process output, or other process variables can be specified as a part of the optimization problem:

$$\begin{array}{rcccl} \mathbf{u}^{\min} & \leq & \mathbf{u} & \leq & \mathbf{u}^{\max}, \\ \Delta\mathbf{u}^{\min} & \leq & \Delta\mathbf{u} & \leq & \Delta\mathbf{u}^{\max}, \\ \mathbf{y}^{\min} & \leq & \hat{\mathbf{y}} & \leq & \mathbf{y}^{\max}, \\ \Delta\mathbf{y}^{\min} & \leq & \Delta\hat{\mathbf{y}} & \leq & \Delta\mathbf{y}^{\max}. \end{array} \tag{8.32}$$

The variables denoted by upper indices min and max are the lower and upper bounds on the signals, respectively.

8.2.3. Receding Horizon Principle

Only the control signal $u(k)$ is applied to the process. At the next sampling instant, the process output $y(k+1)$ is available and the optimization and prediction can be repeated with the updated values. This is called the receding horizon principle. The control action $u(k+1)$ computed at time step $k+1$ will be generally different from the one calculated at time step k, since more up-to-date information about the process is available. This concept is similar to the open-loop control strategy discussed in Section 8.1. Also here, the model can be used independently of the process, in a pure open-loop setting.

A neural or fuzzy model acting as a numerical predictor of the process' output can be directly integrated in the MBPC scheme shown in Figure 8.8. The IMC scheme is usually employed to compensate for the disturbances and modeling errors, see Section 8.1.5.

8.2.4. Optimization in MBPC

The optimization of (8.31) generally requires nonlinear (non-convex) optimization. The following main approaches can be distinguished.

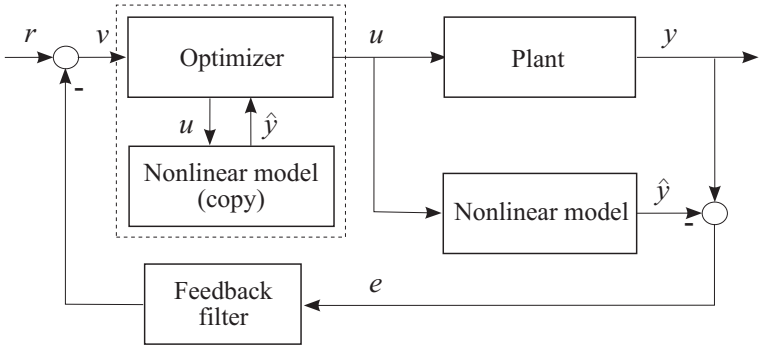


Figure 8.8.: A nonlinear model in the MBPC scheme with an internal model and a feedback to compensate for disturbances and modeling errors.

Iterative Optimization Algorithms

This approach includes methods such as the Nelder-Mead method or sequential quadratic programming (SQP). For longer control horizons (H_c), these algorithms usually converge to local minima. This result in poor solutions of the optimization problem and consequently poor performance of the predictive controller. A partial remedy is to find a good initial solution, for instance, by grid search (Fischer and Isermann, 1998). This is, however, only efficient for small-size problems.

Linearization Techniques

A viable approach to NPC is to linearize the nonlinear model at each sampling instant and use the linearized model in a standard predictive control scheme (Mutha et al., 1997; Roubos et al., 1999). Depending on the particular linearization method, several approaches can be used:

- *Single-Step Linearization.* The nonlinear model is linearized at the current time step k and the obtained linear model is used through the entire prediction horizon. This method is straightforward and fast in its implementation. However, for highly nonlinear processes in conjunction with long prediction horizons, the single-step linearization may give poor results. This deficiency can be remedied by multi-step linearization.

- *Multi-Step Linearization* The nonlinear model is first linearized at the current time step k. The obtained control input $\mathbf{u}(k)$ is then used to predict $\hat{\mathbf{y}}(k+1)$ and the nonlinear model is then again linearized around this future operating point. This procedure is repeated until $k + H_p$. In this way, a more accurate approximation of the nonlinear model is obtained, which is especially useful for longer horizons. The cost one has to pay are increased computational costs.

  For both the single-step and multi-step linearization, a correction step can be employed by using a disturbance vector (Peterson et al., 1992). For the linearized model, the optimal solution of (8.31) is found by the following quadratic program:

$$\min_{\Delta\mathbf{u}} \left\{ \frac{1}{2} \Delta\mathbf{u}^T \mathbf{H} \Delta\mathbf{u} + \mathbf{c}^T \Delta\mathbf{u} \right\}, \tag{8.33}$$

where:

$$\begin{cases} \mathbf{H} = 2({\mathbf{R}_u}^T \mathbf{P} \mathbf{R}_u + \mathbf{Q}) \\ \mathbf{c} = 2[{\mathbf{R}_u}^T \mathbf{P}^T (\mathbf{R}_x \mathbf{A}\mathbf{x}(k) - \mathbf{r} + \mathbf{d})]^T . \end{cases} \tag{8.34}$$

Matrices $\mathbf{R}_u$, $\mathbf{R}_x$ and $\mathbf{P}$ are constructed from the matrices of the linearized system and from the description of the constraints. The disturbance $\mathbf{d}$ can account for the linearization error when it is computed as a difference between the output of the nonlinear model and the linearized model.

- *Feedback Linearization* Also feedback linearization techniques (exact or approximate) can be used within NPC. There are two main differences between feedback linearization and the two operating-point linearization methods described above:
  - The feedback-linearized process has time-invariant dynamics. This is not the case with the process linearized at operating points. Thus, for the latter one, the tuning of the predictive controller may be more difficult.
  - Feedback linearization transforms input constraints in a nonlinear way. This is a clear disadvantage, as the quadratic program (8.33) requires linear constraints. Some solutions to this problem have been suggested (Oliveira et al., 1995; Botto et al., 1996).

- *Discrete Search Techniques* Another approach which has been used to address the optimization in NPC is based on discrete search techniques such as dynamic programming (DP), branch-and-bound (B&B) methods (Lawler and Wood, 1966; Sousa et al., 1997), genetic algorithms (GAs) (Onnen et al., 1997), etc. The basic idea is to discretize the space of the control inputs and to use a smart search technique to find a (near) global optimum in this space. Figure 8.9 illustrates the basic idea of these techniques for the control space discretized into N alternatives: $\mathbf{u}(k+i-1) \in \{\omega_j \mid j = 1, 2, \ldots, N\}$.

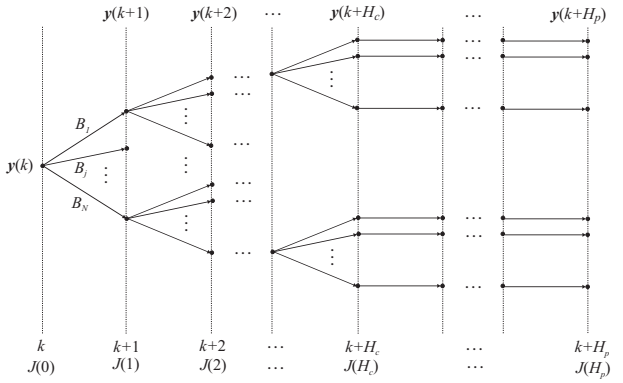
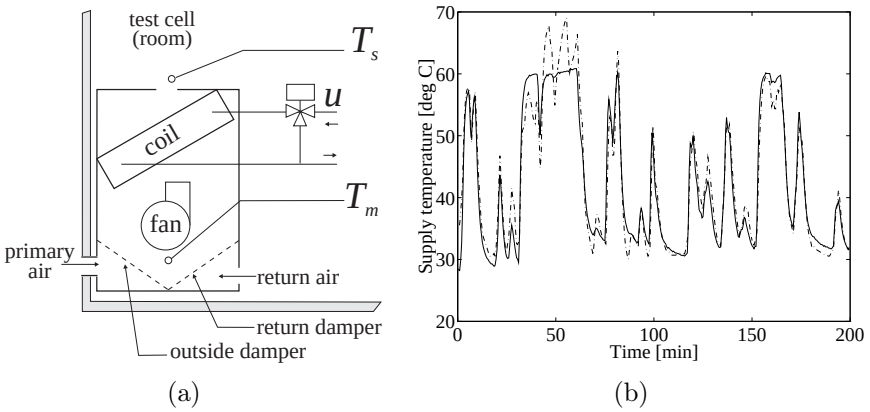


Figure 8.9.: Tree-search optimization applied to predictive control.

It is clear that the number of possible solutions grows exponentially with H_c and with the number of control alternatives. To avoid the search of this huge space, various "tricks" are employed by the different methods. Dynamic programming relies on storing the intermediate optimal solutions in memory. The B&B methods use upper and lower bounds on the solution in order to cut branches that certainly do not lead to an optimal solution. Genetic algorithms search the space in a randomized manner.

Example 8.3 (Control of an Air-Conditioning Unit) Nonlinear predictive control of temperature in an air-conditioning system (Sousa et al., 1997) is shown here as an example. A nonlinear predictive controller has been developed for the control of temperature in a fan coil, which is a part of an air-conditioning system. Hot or cold water is supplied to the coil through a valve. In the unit, outside air is mixed with return air from the room. The mixed air is blown by the fan through the coil where it is heated up or cooled down (Figure 8.10(a)).



(a) (b)

Figure 8.10.: The air-conditioning unit (a) and validation of the TS model (solid line – measured output, dashed line – model output).

This process is highly nonlinear (mainly due to the valve characteristics) and is difficult to model in a mechanistic way. Using nonlinear identification, however, a reasonably accurate model can be obtained within a short time. In the study reported in (Sousa et al., 1997), a TS fuzzy model was constructed from process measurements by means of fuzzy clustering. The obtained model predicts the supply temperature $\mathrm{T_s}$ by using rules of the form:

$$\textbf{If } \hat{\mathrm{T}}_\mathrm{s}(k) \text{ is } A_{i1} \textbf{ and } \mathrm{T_m}(k) \text{ is } A_{i2} \textbf{ and } u(k) \text{ is } A_{13} \textbf{ and } u(k-1) \text{ is } A_{14}$$
$$\textbf{then } \hat{\mathrm{T}}_\mathrm{s}(k+1) = \mathbf{a}_i^T [\hat{\mathrm{T}}_\mathrm{s}(k)\ \mathrm{T_m}(k)\ u(k)\ u(k-1)]^T + b_i$$

The identification data set contained 800 samples, collected at two different times of day (morning and afternoon). A sampling period of 30s was used. The excitation signal consisted of a multi-sinusoidal signal with five different frequencies and amplitudes, and of pulses with random amplitude and width. A separate data set, which was measured on another day, is used to validate the model. Figure 8.10(b) compares the supply temperature measured and recursively predicted by the model.

A model-based predictive controller was designed which uses the B&B method. The controller uses the IMC scheme of Figure 8.11 to compensate for modeling errors and disturbances. The controller's inputs are the setpoint, the predicted supply temperature $\hat{T}_s$, and the filtered mixed-air temperature T_m. The error signal, $e(k) = T_s(k) - \hat{T}_s(k)$, is passed through a first-order low-pass digital filter F_1. A similar filter F_2 is used to filter T_m. Both filters were designed as Butterworth filters, the cut-off frequency was adjusted

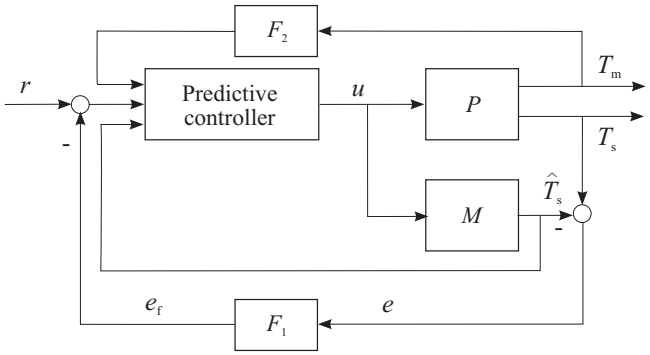


Figure 8.11.: Implementation of the fuzzy predictive control scheme for the fan-coil using an IMC structure.

empirically, based on simulations, in order to reliably filter out the measurement noise, and to provide fast responses.

Figure 8.12 shows some real-time results obtained for $H_c = 2$ and $H_p = 4$.

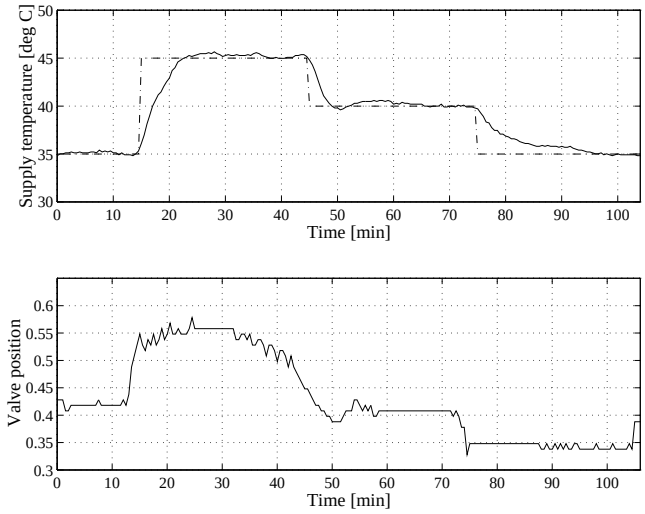


Figure 8.12.: Real-time response of the air-conditioning system. Solid line – measured output, dashed line – reference.

□

8.3. Adaptive Control

Processes whose behavior changes in time cannot be sufficiently well controlled by controllers with fixed parameters. *Adaptive control* is an approach where the controller's parameters are tuned on-line in order to maintain the required performance despite (unforeseen) changes in the process. There are many different way to design adaptive controllers. They can be divided into two main classes:

- *Indirect adaptive control.* A process model is adapted on-line and the controller's parameters are derived from the parameters of the model.

- *Direct adaptive control.* No model is used, the parameters of the controller are directly adapted.

One particular example of indirect adaptive control is presented below. Reinforcement learning, discussed in the next chapter, is an example of a direct adaptive control method.

Indirect Adaptive Control On-line adaptation can be applied to cope with the mismatch between the process and the model. In many cases, a mismatch occurs as a consequence of (temporary) changes of process parameters. To deal with these phenomena, especially if their effects vary in time, the model can be adapted in the control loop. Since the control actions are derived by inverting the model on line, the controller is adjusted automatically. Figure 8.13 depicts the IMC scheme with on-line adaptation of the consequent parameters in the fuzzy model.

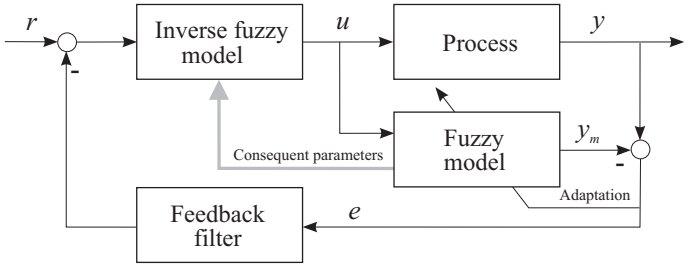


Figure 8.13.: Adaptive model-based control scheme.

Since the model output given by eq. (8.17) is linear in the consequent parameters, standard *recursive least-squares algorithms* can be applied to estimate the consequent parameters from data. It is assumed that the rules of the fuzzy model are given by (8.13) and the consequent parameters are indexed sequentially by the rule number. The column vector of the consequents is denoted by $\mathbf{c}(k) = [c_1(k), c_2(k), \ldots, c_K(k)]^T$, where K is the number of rules. The normalized degrees of fulfillment denoted by $\boldsymbol{\gamma}_i(k)$ are computed by:

$$\gamma_i(k) = \frac{\beta_i(k)}{\sum_{j=1}^{K} \beta_j(k)}, \quad i = 1, 2, \ldots, K\,. \tag{8.35}$$

They are arranged in a column vector $\boldsymbol{\gamma}(k) = [\gamma_1(k), \gamma_2(k), \ldots, \gamma_K(k)]^T$. The consequent vector $\mathbf{c}(k)$ is updated recursively by:

$$\mathbf{c}(k) = \mathbf{c}(k-1) + \frac{\mathbf{P}(k-1)\boldsymbol{\gamma}(k)}{\lambda + \boldsymbol{\gamma}^T(k)\mathbf{P}(k-1)\boldsymbol{\gamma}(k)} \left[y(k) - \boldsymbol{\gamma}^T(k)\mathbf{c}(k-1)\right], \tag{8.36}$$

where λ is a constant forgetting factor, which influences the tracking capabilities of the adaptation algorithm. The smaller the λ, the faster the consequent parameters adapt, but the algorithm is more sensitive to noise. Therefore, the choice of λ is problem dependent. The covariance matrix $\mathbf{P}(k)$ is updated as follows:

$$\mathbf{P}(k) = \frac{1}{\lambda}\left[\mathbf{P}(k-1) - \frac{\mathbf{P}(k-1)\boldsymbol{\gamma}(k)\boldsymbol{\gamma}^T(k)\mathbf{P}(k-1)}{\lambda + \boldsymbol{\gamma}^T(k)\mathbf{P}(k-1)\boldsymbol{\beta}(k)}\right]. \tag{8.37}$$

The initial covariance is usually set to $\mathbf{P}(0) = \alpha \cdot \mathbf{I}$, where $\mathbf{I}$ is a $K \times K$ identity matrix and α is a large positive constant.

8.4. Summary and Concluding Remarks

Several methods to develop nonlinear controllers that are based on an available fuzzy or neural model of the process under consideration have been presented. They include inverse model control, predictive control and two adaptive control techniques. Internal model control scheme can be used as a general method for rejecting output-additive disturbances and minor modeling errors in inverse and predictive control.

8.5. Problems

1. Draw a general scheme of a feedforward control scheme where the controller is based on an inverse model of the dynamic process. Describe the blocks and signals in the scheme.

2. Consider a first-order affine Takagi–Sugeno model:

$$R_i \qquad \textbf{If } y(k) \text{ is } A_i \textbf{ then } y(k+1) = a_i y(k) + b_i u(k) + c_i$$

   Derive the formula for the controller based on the inverse of this model, i.e., $u(k) = f(r(k+1), y(k))$, where r is the reference to be followed.

3. Explain the concept of predictive control. Give a formula for a typical cost function and explain all the symbols.

4. What is the principle of indirect adaptive control? Draw a block diagram of a typical indirect control scheme and explain the functions of all the blocks.

5. Explain the idea of internal model control (IMC).

9. Reinforcement Learning

9.1. Introduction

Reinforcement Learning (RL) is a method of machine learning, which is able to solve complex optimization and control tasks in an interactive manner. In the RL setting, there is an *agent* and an *environment*, which are explicitly separated one from another. The environment represents the system in which a task is defined. For example, the environment can be a maze, in which the task is to find the shortest path to the exit. The agent is a *decision maker*, whose goal it is to accomplish the task.

In RL this problem is solved by letting the agent interact with the environment. Each *action* of the agent changes the state of the environment. The environment responds by giving the agent a *reward* for what it has done. The value of this reward indicates to the agent how good or how bad its action was in that particular state. Based on this reward, the agent *adapts* its behavior. The agent then observes the state of the environment and determines what action it should perform next. In this way, the agent learns to act, such that its reward is maximized. It is obvious that the behavior of the agent strongly depends on the rewards it receives; the problem to be solved is implicitly defined by these rewards.

It is important to notice that contrary to *supervised-learning* (see Chapter 7), there is no teacher that would guide the agent to the goal. Neither is the learning completely unguided, like in *unsupervised-learning* (see Chapter 4). RL is very similar to the way humans and animals learn. By trying out different actions and learning from our mistakes, we are able to solve complex tasks, such as riding a bike, or playing the game of chess. There are examples of tasks that are very hard, or even impossible to be solved by hand-coded software routines. In principle, RL is capable of solving such complex tasks.

Example 9.1 Humans are able optimize their behavior in a particular environment without knowing an accurate model of that environment. Many learning tasks consist of repeated trials followed by a reward or punishment. Each trial can be a dynamic sequence of actions while the performance evaluation (reinforcement) is only received at the end.

Consider, for instance, that you are learning to play tennis. The control trials are your attempts to correctly hit the ball. In supervised learning you would have a teacher who would evaluate your performance from time to time and would tell you how to change your strategy in order to improve yourself. The advise might be very detailed in terms of how to change the grip, how to approach the balls, etc.

In reinforcement learning, on the other hand, the role of the teacher is only to tell you whether a particular shot was OK (reward) or not (punishment). It is left to you to determine the appropriate corrections in your strategy (you would not pay such a teacher, of course).

It is important to realize that each trial can be a dynamic sequence of actions (approach the ball, take a stand, hit the ball) while the actual reinforcement is only received at the end. Therefore, a large number of trials may be needed to figure out which particular actions were correct and which must be adapted.

□

This chapter describes the basic concepts of RL. First, in Section 9.2, we formalize the environment and the agent and discuss the basic mechanisms that underlie the learning. In Section 9.3, reinforcement learning methods for problems where the agent knows a model of the environment are discussed. Section 9.4 deals with problems where such a model is not available to the agent. The important concept of exploration is discussed in greater detail in Section 9.5. Section 9.6 presents a brief overview of some applications of RL and highlights two applications where RL is used to control a dynamic system. Please note that sections denoted with a star * are for the interested reader only and should not be studied by heart for this course.

9.2. The Reinforcement Learning Model

9.2.1. The Environment

In RL, the main components are the agent and the environment. The agent is defined as the learner and decision maker and the environment as everything that is outside of the agent. Let the state of the environment be represented by a state variable $x(t)$ that changes over time. For RL however, the state variable is considered to be observed in *discrete time* only: x_k, for $k = 1, 2 \ldots$.[1] The reason for this is that the most RL algorithms work in discrete time and that the most research has been done for discrete time RL. Furthermore, traditional RL algorithms require the state of the environment to be perceived by the agent as *quantized*. In accordance with the notations in RL literature, this quantized state is denoted as $s_k \in S$, where S is the set of possible states of the environment (state space) and k the index of discrete time.

The state of the environment can be observed by the agent. Based on this observation, the agent determines an action to take. This action alters the state of the environment and gives rise to the reward, which the environment provides to the agent. A distinction is made between the reward that is immediately received at the end of time step k, and the *total reward* received by the agent over a period of time. This *immediate reward* is a scalar denoted by $r_k \in \mathbb{R}$. The total reward is some function of the immediate rewards $R_k = f(\ldots, r_{k-1}, r_k, r_{k+1}, \ldots)$.

9.2.2. The Agent

As has just been mentioned, the agent can influence the state by performing an action for which it receives an immediate reward. This action $a_k \in A$, with A the set of possible actions, is translated to the physical input to the environment u_k. This u_k can be continuous valued, where a_k can only take values from the set A. In the same way, the physical output

[1] To simplify the notation in this chapter, the time index k is typeset as a lower subscript.

of the environment y_k, is observed by the agent as the state s_k. There is a distinction between the cases when this observed state has a one-to-one relation to the actual state of the environment and when it is not. In the latter case, these environments are said to be *partially observable*. RL in partially observable environments is outside the scope of this chapter.

The learning task of the agent is to find an *optimal mapping* from states to actions. This mapping $s_k \rightarrow a_k$ is called the *policy*. A policy that defines a deterministic mapping is denoted by $\pi_k(s)$. A policy that gives a probability distribution over states and actions is denoted as $\Pi_k(s, a)$.

The interaction between the environment and the agent is depicted in Figure 9.1. Here, T is the transition function, modeling the environment, R is the reward function, defining the reward in relation to the state of the environment and q^{-1} is the one-step delay operator. The environment is considered to have the *Markov property*, which is the subject of the next section.

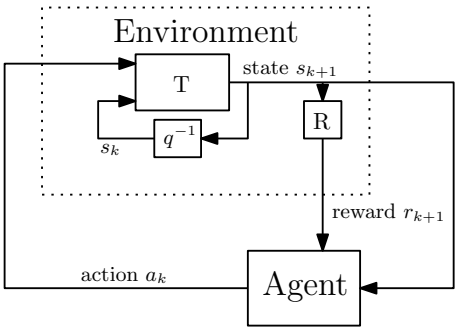


Figure 9.1.: The interaction between the environment and the agent in reinforcement learning.

9.2.3. The Markov Property

The environment is described by a mapping from a state to a new state given an input, $s_{k+1} = f(s_k, a_k)$. Note that both the state s and action a are generally vectors. In RL, it is often easier to regard this mapping as a probability distribution over the current quantized state and the new quantized state given an action. If also the immediate reward is considered that follows after such a transition, the mapping $s_k, a_k \rightarrow s_{k+1}, r_{k+1}$ results. This probability distribution can be denoted as

$$P\{s_{k+1}, r_{k+1} \mid s_k, a_k, r_k, s_{k-1}, a_{k-1}, \ldots, r_1, s_0, a_0\}, \tag{9.1}$$

where $P\{x \mid y\}$ is the probability of x given that y has occurred.

This is the general state, action transition model of the environment that defines the probability of transition to a certain new state s' with a certain immediate reward, given the complete sequence of current state, action and all past states and actions. When the environment is described by a state signal that summarizes all past sensations compactly, yet in such a way that all relevant information is retained, the environment is said to have the *Markov property* (Sutton and Barto, 1998). The probability distribution (9.1) can then

be denoted as

$$P\{s_{k+1}, r_{k+1} \mid s_k, a_k\}. \tag{9.2}$$

An RL task that satisfies the Markov property is called a Markov Decision Process (MDP). In that case, one-step dynamics are sufficient to predict the next state and immediate reward. For any state s, action a and next state s', the state transition probability function

$$\mathcal{P}^a_{ss'} = P\{s_{k+1} = s' \mid s_k = s, a_k = a\} \tag{9.3}$$

and the expected value of the next reward,

$$\mathcal{R}^a_{ss'} = E\{r_{k+1} \mid s_k = s, a_k = a, s_{k+1} = s'\} \tag{9.4}$$

are the two functions that completely specify the most important aspects of a finite MDP.

The MDP assumes that the agent observes the complete state of the environment. In Figure 9.1, this is depicted as the direct link between the state of the environment and the agent. In practice however, this is never possible. First of all, sensor limitations, such as its range, and sensor noise distort the observations. Second, in some tasks, such as an agent navigating its way trough a maze, the agent has to extract in what kind of state it is, such as "a corridor" or "a T-crossing" (Bakker, 2004) based on observations. It can therefore be impossible for the agent to distinguish between two or more possible states. In these cases, the environment may still have the Markov property, but the agent only observes a part of it. An RL task that is based on this assumption is called Partially Observable MDP, or POMDP .

9.2.4. The Reward Function

The task of the agent is to maximize some measure of *long-term reward*. The long-term reward at time step k, also called the *return*, is some monotonically increasing function of all the immediate rewards received after k. With the immediate reward after an action at time step k denoted as r_{k+1}, one way of computing the return R_k is to simply sum all the received immediate rewards following the actions that eventually lead to the goal:

$$R_k = r_{k+1} + r_{k+2} + \ldots + r_N = \sum_{n=0}^{N} r_{n+k+1}. \tag{9.5}$$

This measure of long-term reward assumes a finite horizon; i.e., the goal is reached in at most N steps. The RL task is then called *episodic*, or ending in a *terminal state*. When an RL task is not episodic, it is *continuing* and the sum in eq. (9.5) would become infinite. To prevent this, the rewards that are further in future are discounted:

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \ldots = \sum_{n=0}^{\infty} \gamma^n r_{k+n+1}, \tag{9.6}$$

where $0 \leq \gamma < 1$ is the *discount rate* (Sutton and Barto, 1998).

There are many ways of interpreting γ. One could interpret γ as the probability of living another step, as the uncertainty about future events or as some rate of interest in future events (Kaelbing et al., 1996). Apart from intuitive justification, the dominant reason of

using a discount rate is that it mathematically bounds the sum: when $\gamma < 1$ and $\{r_k\}$ is bounded, R_k is finite (Sutton and Barto, 1998). Also for episodic tasks, discounting future rewards is a good idea for the same (intuitive) reasons as mentioned above. The expression for the return is then

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \ldots + \gamma^{K-1} r_{k+K} = \sum_{n=0}^{K-1} \gamma^n r_{k+n+1}. \tag{9.7}$$

The next section shows how the return can be processed to make decisions about which action the agent should take in a certain state.

9.2.5. The Value Function

In the previous section, the return was expressed as some combination of the *future* immediate rewards. Naturally, the agent operating in a causal environment does not know these future rewards in advance. Accordingly, it does not know for sure if a chosen policy will indeed result in maximizing the return. The agent can, however, determine its policy based on the *expected return*. *Value functions* define for each state or state-action pair the expected return and thus express how good it is to be in that state following policy π, or to perform a certain action in that state. The *state-value function* for an agent in state s following a deterministic policy π is defined for discounted rewards as:

$$V^\pi(s) = E^\pi\{R_k \mid s_k = s\} = E^\pi\{\sum_{n=0}^{\infty} \gamma^n r_{k+n+1} \mid s_k = s\}, \tag{9.8}$$

where $E^\pi\{\ldots\}$ denotes the expected value of its argument given that the agent follows a policy π. In a similar manner, the *action-value function* for an agent in state s performing an action a and following the policy π afterwards is defined as:

$$Q^\pi(s,a) = E^\pi\{R_k \mid s_k = s, a_k = a\} = E^\pi\{\sum_{n=0}^{\infty} \gamma^n r_{k+n+1} \mid s_k = s, a_k = a\}. \tag{9.9}$$

Now, the notion of *optimal policy* π^* can be formalized as the policy that maximizes the value functions. Denote by $V^*(s)$ and $Q^*(s,a)$ the optimal state-value and action-value function in the state s respectively:

$$V^*(s) = \max_\pi V^\pi(s) \tag{9.10}$$

and

$$Q^*(s,a) = \max_\pi Q^\pi(s,a). \tag{9.11}$$

In RL methods and algorithms, either $V(s)$ or $Q(s,a)$ is used. They are never used simultaneously. The optimal policy π^* is defined as the argument that maximizes either eq. (9.10) or (9.11). The agent has a set of parameters, that keeps track of these state-values, or action-values in each state. In classical RL, this set of parameters is a *table*, in which every state or state-action pair is associated with a value. Knowing the value functions, the task of the agent in RL can be described as learning a policy that maximizes the state-value function or the action-value function.

9.2.6. The Policy

The policy has already been defined as a mapping from states to actions ($s_k \rightarrow a_k$). Two types of policy have been distinguished, namely the deterministic policy $\pi_k(s)$ and the stochastic policy $\Pi_k(s, a)$. In most RL methods, the policy is deterministic. The most straightforward way of determining the policy is to assign the action to each state that maximizes the action-value, also called Q-value. This is called the *greedy policy*. By always following a greedy policy during learning, it is highly unlikely for the agent to find the optimal solution to the RL task. During learning, the agent can also follow a non-greedy policy. This is called *exploration*. In this way, it is able to explore parts of its environment where potentially better solutions to the problem may be found. This is the subject of Section 9.5. When the learning has converged to the optimal value function, the solution is presented in terms of a greedy policy:

$$\pi(s) = \arg\max_{a' \in A} Q^*(s, a'). \tag{9.12}$$

The stochastic policy is actually a mapping from state-action pairs to a probability that this action will be chosen in that state. Effectively, when only the *outcome* of this policy is regarded, there is again a mapping from states to actions. It is important to remember that this mapping is not fixed. Each time the policy is called, it can output a different action, because of the probability distribution underlying it.

An example of a stochastic policy is *reinforcement comparison* (Sutton and Barto, 1998). This method differs from the methods that are discussed in this chapter, because it does not use value functions. It maintains a measure for the preference of a certain action in each state, $p_k(s, a)$. This preference is updated at each time step according to the deviation of the immediate reward w.r.t. the average of the reward received in that state so far $\bar{r}_k(s)$.

$$\bar{r}_{k+1}(s) = \bar{r}_k(s) + \alpha(r_k(s) - \bar{r}_k(s)), \tag{9.13}$$

where $0 < \alpha \leq 1$ is a static *learning rate*.

This updated average reward is used to determine the way in which the preferences should be adapted.

$$p_{k+1}(s, a) = p_k(s, a) + \beta(r_k(s) - \bar{r}_k(s)), \tag{9.14}$$

with β a positive step size parameter.

The policy is some function of these preferences, like e.g.

$$\Pi_k(s, a) = \frac{e^{p_k(s,a)}}{\sum_b e^{p_k(s,b)}}. \tag{9.15}$$

There exist methods to determine policies that are effectively in between stochastic and deterministic policies. E.g. in *pursuit methods* (Sutton and Barto, 1998), the policy is stochastic during learning, but converges to a deterministic policy towards the end of the learning. These methods will not be discussed here any further. In the sequel, we focus on deterministic policies.

The next sections discuss reinforcement learning for both the case when the agent has no model of the environment and when it has.

9.3. Model Based Reinforcement Learning

Reinforcement learning methods assume that the environment can be described by a Markovian decision process (MDP). A process is an MDP when the next state of the environment only depends on its current state and input (see eq. (9.2)). Here it is assumed that the complete state can be observed by the agent. When this is not the case, the process is called an POMDP. In this section, RL methods are discussed that assume an MDP with a known model (i.e. a known state transition, and reward function).

9.3.1. Bellman Optimality

Solution techniques for MDP that use an exact model of the environment are known as dynamic programming (DP). DP is based on the *Bellman equations* that give recursive relations of the value functions. For state-values this corresponds to

$$\begin{aligned} V^\pi(s) &= E^\pi\left\{\sum_{n=0}^{\infty}\gamma^n r_{k+n+1} \mid s_k = s\right\} && (9.16)\\ &= E^\pi\left\{r_{k+1} + \gamma\sum_{n=0}^{\infty}\gamma^n r_{k+n+2} \mid s_k = s'\right\} && (9.17)\\ &= \sum_a \Pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma E^\pi\left\{\sum_{n=0}^{\infty}\gamma^n r_{k+n+2} \mid s_k = s'\right\}\right] && (9.18)\\ &= \sum_a \Pi(s,a)\sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right], && (9.19) \end{aligned}$$

with $\Pi(s,a)$ the probability of performing action a in state s while following policy π, and $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ the state-transition probability function (9.3) and the expected value of the next immediate reward (9.4), respectively.

With (9.16) substituted in (9.10) and (9.12) an expression called the *Bellman optimality equation* can be derived (Sutton and Barto, 1998):

$$V^*(s) = \max_a \sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right], \tag{9.20}$$

for state-values where s' denotes the next state and

$$Q^*(s,a) = \sum_{s'}\mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma\max_{a'}Q^*(s',a')\right], \tag{9.21}$$

for action-values.

Eq. (9.20) is a system of n equations in n unknowns, when n is the dimension of the state-space. Eq. (9.21) is a system of $n \times n_a$ equations in $n \times n_a$ unknowns, where A is the number of actions in the action set. One can solve these equations when the dynamics of the environment are known ($\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$). This corresponds to knowing the world model a priori. When either $V^*(s)$ or $Q^*(s,a)$ has been found, the optimal policy simply consists in taking the action a in each state s encountered for which $Q^*(s,a)$ is the highest or results in the highest expected direct reward plus discounted state value $V^*(s)$. The beauty is that

a one-step-ahead search yields the long-term optimal actions, since $V^*(s')$ incorporates the expected long-term reward in a local and immediate quantity. The next two sections present two methods for solving (9.20), called *policy iteration* and *value iteration*.

9.3.2. Policy Iteration

Policy iteration consists of alternately performing a *policy evaluation* and a *policy improvement* step. In the policy evaluation step, the algorithm starts with the current policy and computes its value function by iterating the following update rule:

$$V_{n+1}(s) = E^{\pi}\left\{r_{k+1} + \gamma V_n(s_{k+1}) \mid s_k = s\right\} \tag{9.22}$$

$$= \sum_a \Pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_n(s')\right], \tag{9.23}$$

where π denotes the deterministic policy that is followed. $\Pi(s,a)$ is a matrix with $\Pi(s,\pi(s)) = 1$ and zeros elsewhere.

This equation is an update rule derived from the Bellman equation for V^π (9.16). The sequence $\{V_n\}$ is guaranteed to converge to V^π as $n \to \infty$ and $\gamma < 1$. For episodic tasks, γ can also be 1 (Sutton and Barto, 1998).

When V^π has been found, it is used to improve the policy. For this, the action value function for the current policy, $Q^\pi(s,a)$ has to be computed from V^π in a similar way as in eq. (9.21). The new policy is computed greedily:

$$\pi'(s) = \arg\max_a Q^\pi(s,a) \tag{9.24}$$

$$= \arg\max_a E\left\{r_{k+1} + \gamma V^\pi(s_{k+1}) \mid s_k = s, a_k = a\right\} \tag{9.25}$$

$$= \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]. \tag{9.26}$$

Sutton and Barto (1998) show in the *policy improvement theorem* that the new policy π' is better or as good as the old policy π. From alternating between policy evaluation and policy improvement steps (9.23) and (9.26) respectively, the optimal value function V^* and the optimal policy π^* are guaranteed to result. The general case of policy iteration, called *general policy iteration* is depicted in Figure 9.2.

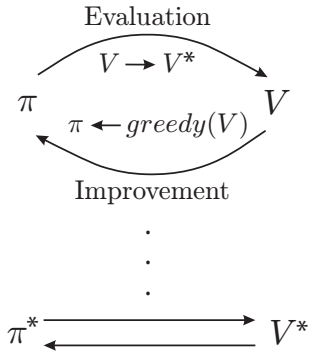


Figure 9.2.: General Policy Iteration.

Because the best policy resulting from this value function can already be the optimal policy even when V^n has not fully converged to V^π, it is wise to stop when the difference between two consecutive updates is smaller than some specific threshold ϵ, also called the *Bellman residual*. When choosing this bound wisely, this process may still result in an optimal policy, but much faster than without the bound. The resulting non-optimal V^π is then bounded by (Kaelbing et al., 1996):

$$V^\pi(s) \geq V^*(s) - \frac{2\gamma\epsilon}{1-\gamma}. \tag{9.27}$$

Policy iteration can converge faster to the optimal policy and value function when policy improvement starts with the value function of the previous policy. It uses only a few evaluation- improvement iterations to converge. A drawback is that it requires many sweeps over the complete state-space, which is computationally very expensive. A method that is much faster per iteration, but requires more iterations to converge is *value iteration*. This is the subject of the next section.

9.3.3. Value Iteration

Value iteration performs only one sweep over the state-space for the current optimal policy in the policy evaluation step. Instead of iterating until V_n converges to V^π, the process is truncated after one update. It uses the Bellman optimality equation (9.20) as an update rule:

$$V_{n+1}(s) = \max_a E\left\{r_{k+1} + \gamma V_n(s_{k+1}) \mid s_k = s, a_k = a\right\} \tag{9.28}$$

$$= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_n(s')\right]. \tag{9.29}$$

The policy improvement step is the same as with policy iteration. The iteration is stopped when the change in value function is smaller than some specific ϵ. In some situations, ϵ has to be very small to get to the optimal policy. In these cases, value iteration needs many iterations. According to (Wiering, 1999), value iteration outperforms policy iteration in terms of number of iterations needed.

9.4. Model Free Reinforcement Learning

In the previous section, methods have been discussed for solving the reinforcement learning task. These methods assume that there is an explicit model of the environment available for the agent. In real-world application, however, the agent typically has no such model. A way for the agent to solve the problem, is to first learn a model of the environment and then apply the methods from the previous section, like value iteration. Another way is to use what are called emphtemporal difference methods. These RL methods do not require a model of the environment to be available to the agent. Model free reinforcement learning is the subject of this section.

9.4.1. The Reinforcement Learning Task

Before discussing temporal difference methods, it is important to know the structure of any RL task. A schematic is depicted in Figure 9.3.

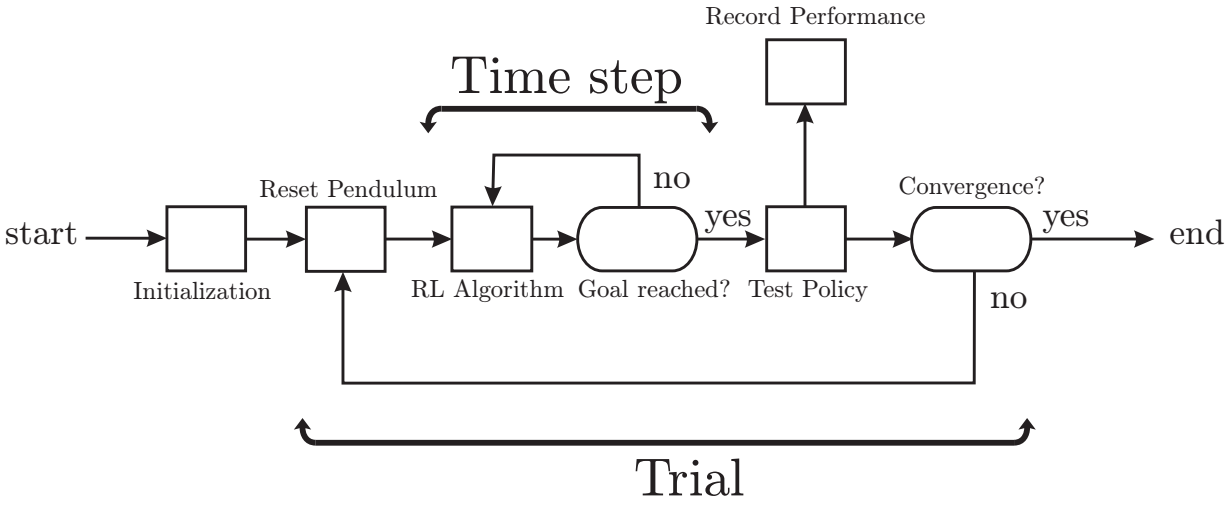


Figure 9.3.: The reinforcement learning scheme.

At the start of learning, all parameters and value functions are initialized. There are a number of trials or episodes in which the learning takes place. In general, one speaks about trials. Only when the trial explicitly ends in a terminal state, the trial can also be called an episode. In each trial, the learning consists of a number of time steps. An episode ends when one of the terminal states has been reached. The policy that led to this state is evaluated. Whenever it is determined that the learning has converged, the learning is stopped. The learning can also be stopped when the number of trials exceeds a certain maximum, or when there is no change in the policy for some number of times in a row.

9.4.2. Temporal Difference

In Temporal Difference (TD) learning methods (Sutton and Barto, 1998), the agent processes the immediate rewards it receives at each time step, thereby learning from each action. A learning rate (α) determines the importance of the new estimate of the value function over the old estimate. The simplest TD update rule is as follows:

$$V(s_k) \leftarrow (1-\alpha_k)V(s_k) + \alpha_k \left[r_{k+1} + \gamma V(s_{k+1})\right], \tag{9.30}$$

or put in a different way as:

$$V(s_k) \leftarrow V(s_k) + \alpha_k \left[r_{k+1} + \gamma V(s_{k+1}) - V(s_k)\right], \tag{9.31}$$

where $\alpha_k(a)$ is the learning-rate used to process the reward received after the k^{th} time step. From the second expression, the TD-error is the part between the brackets. Here, the learning rate determines how strong the TD-error determines the new prediction of the value function $V(s_k)$. In the limit, V will converge to the value-function belonging to

the optimal policy $V^{\pi*}$ when α is annealed properly. Convergence is guaranteed when α_k satisfies the conditions

$$\sum_{k=1}^{\infty} \alpha_k = \infty \tag{9.32}$$

and

$$\sum_{k=1}^{\infty} \alpha_k^2 < \infty, \tag{9.33}$$

With $\alpha_k = \alpha$ a constant learning rate, the second condition is not met. In that case, the value function constantly changes when new rewards are received. For nonstationary problems, this is desirable. For stationary problems, this choice of learning rate might still results in the agent to find the optimal policy, as α is chosen small enough that it still changes the nearly converged value function, but does not affect the policy. The first condition is to ensure that the learning rate is large enough to overcome initial conditions or random fluctuations (Sutton and Barto, 1998). There are no general rules for choosing an appropriate learning-rate in a certain application.

With the most basic TD update (9.31), $V(s_k)$ is only updated after receiving the new reward r_{k+1}. At the next time step, the agent is in the state s_{k+1}. When it performs an action for which it receives an immediate reward r_{k+2}, this reward in turn is only used to update $V(s_{k+1})$. It would be reasonable to also update $V(s_k)$, since this state was also responsible for receiving the reward r_{k+2} two time steps later. The same can be said about all other state-values preceding $V(s_k)$. It is reasonable to say that immediate rewards should change the values in all states that were visited in order to get to the current state. Wisely, states further away in the past should be credited less than states that occurred more recently. How the credit for a reward should best be distributed is called *the credit assignment problem*. A basic method in reinforcement learning is to combine the so called *eligibility traces* (Sutton and Barto, 1998) with TD-learning. This is the subject of the next section.

9.4.3. Eligibility Traces

Eligibility traces mark the parameters that are responsible for the current event and therefore eligible for chance. An extra variable, called the *eligibility trace* is associated with each state, indicating how eligible it is for a change when a new reinforcing event comes available. The eligibility trace for state s at discrete time k is denoted as $e_k(s) \in \mathbb{R}^+$. At each time step in the trial, the eligibility traces for all states decay by a factor $\gamma\lambda$, where γ is still the discount rate and λ is a new parameter, called the *trace-decay parameter* (Sutton and Barto, 1998). The eligibility trace for the state just visited can be incremented by 1,

$$e_k(s) = \begin{cases} \gamma\lambda e_{k-1}(s) & \text{if } s \neq s_k \\ \gamma\lambda e_{k-1}(s) + 1 & \text{if } s = s_k \end{cases}, \tag{9.34}$$

for all nonterminal states s, which is called *accumulating traces*. Another way of changing the trace in the state just visited is to set it to 1,

$$e_k(s) = \begin{cases} \gamma\lambda e_{k-1}(s) & \text{if } s \neq s_k \\ 1 & \text{if } s = s_k \end{cases}, \tag{9.35}$$

which is called *replacing traces*. The method of accumulating traces is known to suffer from convergence problems, therefore replacing traces is used almost always in the literature to update the eligibility traces.

The reinforcement event that leads to updating the values of the states is the 1-step TD error,

$$\delta_k = r_{k+1} + \gamma V_k(s_{k+1}) - V_k(s_k). \tag{9.36}$$

The elements of the value function (9.31) are then updated by an amount of

$$\Delta V_k(s) = \alpha \delta_k e_k(s), \tag{9.37}$$

for all s.

Eligibility traces bridge the gap from single-step TD to methods that wait until the end of the episode when all the reward has been received and the values of the states can be updated all together. Such methods are known as *Monte Carlo* (MC) methods. These methods are not very efficient in the way they process information. During an episode, the agent is performing guesses on what action to take in each state [2]. It is blind for the immediate rewards received during the episode. The value of λ in TD(λ) determines the trade-off between MC methods and 1-step TD. In particular TD(0) is the 1-step TD and TD(1) is MC learning.

Three particular popular implementations of TD are Watkins' Q-learning, SARSA and actor-critic RL. The next three sections discuss these RL algorithms.

9.4.4. Q-learning

Basic TD learning learns the state-values, rather than action-values. As discussed in Section 9.2.6, action-values are used to derive the greedy actions directly. Deriving the greedy actions from a state-value function is computationally more expensive. This advantage of action-values comes with a higher memory consumption, as values need to be stored for all state-action pairs instead of only for all states. Still for control applications in particular, action-values are much more convenient than state-values. An algorithm that learns Q-values is Watkins' Q-learning (Watkins and Dayan, 1992). This algorithm is a so called *off-policy* TD control algorithm as it learns the action-values that are not necessarily on the policy that it is following. In its simplest form, *1-step Q-learning* is defined by

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha \left[r_{k+1} + \gamma \max_a Q(s_{k+1}, a) - Q(s_k, a_k) \right]. \tag{9.38}$$

Thus, in the next state s_{k+1}, the maximum action value is used independently of the policy actually followed. To guarantee convergence, all state-action pairs need to be visited continually. This is a general assumption for all reinforcement learning methods.

In order to incorporate eligibility traces with Q-learning, a trace should be associated with each state-action pair rather than with states as in TD(λ)-learning. For the off-policy nature of Q-learning, incorporating eligibility traces in Q-learning needs special attention. State-action pairs are only eligible for change when they are followed by greedy actions.

[2] Hence the name Monte Carlo. It resembles the way gamblers play in the casino's of which this capital of Monaco is famous for.

Eligibility traces, thus only work up to the moment when an explorative action is taken. Whenever this happens, the eligibility trace is reset to zero. The algorithm is given as:

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha\delta_k e_k(s,a), \quad \forall s,a \tag{9.39}$$

where

$$\delta_k = r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \tag{9.40}$$

Clearly, cutting of the traces every time an explorative action is chosen, takes away much of the advantages of using eligibility traces. Especially when explorative actions are frequent, as it is in early trials, $Q(\lambda)$-learning will not be much faster than regular Q-learning. An algorithm that combines eligibility traces more efficiently with Q-learning is Peng's $Q(\lambda)$, however its implementation is much more complex compared to the implementation of Watkins' $Q(\lambda)$. For this reason, Peng's $Q(\lambda)$ will not be described. According to Sutton and Barto (1998), the development of $Q(\lambda)$-learning has been one of the most important breakthroughs in RL.

9.4.5. SARSA

Like Q-learning, SARSA learns state-action values rather than state values. The *1-step SARSA* update rule is defined as:

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha \left[r_{k+1} + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k) \right]. \tag{9.41}$$

The SARSA update needs information about the current and next state-action pair. It thus updates over the policy that it decides to take. For this reason, as opposed to Q-learning, SARSA is an on-policy method. In a similar way as with Q-learning, eligibility traces can be incorporated to SARSA. However, as SARSA is an on-policy method, the trace does not need to be reset to zero when an explorative action is taken.

9.4.6. Actor-Critic Methods

The actor-critic RL methods have been introduced to deal with continuous state and continuous action spaces (recall that Q-learning works for a discrete MDP). The value function and the policy are separated. The value function is represented by the so-called *critic*. The role of the critic is to predict the outcome of a particular control action in a particular state of the process.

The control policy is represented separately from the critic and is adapted by comparing the reward actually received to the one predicted by the critic. A block diagram of a classical actor-critic scheme (Barto et al., 1983; Anderson, 1987) is depicted in Figure 9.4. It consists of a reward, the critic and the actor, which are detailed in the subsequent sections.

Reward

This block provides the reward function r_k, also called *external reinforcement*.

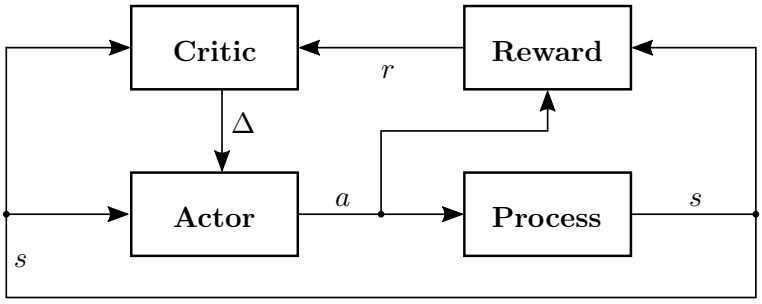


Figure 9.4.: The actor-critic learning scheme.

Critic

The task of the critic is to predict the expected future reinforcement r the process will receive being in the current state and following the current control policy. This prediction is then used to obtain a more informative signal, called the *internal reinforcement*, which is involved in the adaptation of the critic and the actor. The critic is trained to predict the future value function $V(s_k)$ for the current process state s_k. Denote $\hat{V}(s_k)$ the prediction of $V(s_k)$. To derive the adaptation law for the critic, we rewrite the equation for the value function as:

$$V(s_k) = \sum_{n=0}^{\infty} \gamma^n r_{k+n+1} = r_{k+1} + \gamma V(s_{k+1})\,. \tag{9.42}$$

To train the critic, we need to compute its prediction error $\Delta_k = V(s_k) - \hat{V}(s_k)$. The true value function $V(s_k)$ is unknown, but it can be approximated by replacing $V(s_{k+1})$ in (9.42) by its prediction $\hat{V}(s_{k+1})$. This gives an estimate of the prediction error:

$$\Delta_k = V(s_k) - \hat{V}(s_k) = r_{k+1} + \gamma \hat{V}(s_{k+1}) - \hat{V}(s_k)\,. \tag{9.43}$$

As Δ_k is computed using two consecutive values $\hat{V}(s_k)$ and $\hat{V}(s_{k+1})$, and is the *temporal difference* that was already described in Section 9.4.2. Note that both $\hat{V}(s_k)$ and $\hat{V}(s_{k+1})$ are known at time k, since $\hat{V}(s_{k+1})$ is a prediction obtained for the current process state. The temporal difference error serves as the internal reinforcement signal, see Figure 9.4. The temporal difference can be directly used to adapt the critic. Let the critic be represented by a neural network or a fuzzy system

$$\hat{V}(s_{k+1}) = \hat{V}(s_{k+1}, \theta_k), \tag{9.44}$$

where θ_k is a vector of adjustable parameters. To update θ_k, a gradient-descent learning rule is used:

$$\theta_{k+1} = \theta_k + \alpha_c \Delta_k \frac{\partial \hat{V}(s_k, \theta_k)}{\partial \theta_k}, \tag{9.45}$$

where $\alpha_c > 0$ is the critic's learning rate.

Actor

When the critic is trained to predict the future system's performance (the value function), the actor (i.e., the policy) can be adapted in order to establish an optimal mapping between

the system states and the control actions. The temporal difference is used to adapt the actor as follows:

Given a certain state s_k, the control action a_k^π is calculated using the current policy $\pi(s_k)$. This action is not applied to the process, but it is modified to obtain a_k by adding exploration $\tilde{a}_k$ to it. The exploration $\tilde{a}_k$ can, for example, be a random value from $\mathrm{N}(0, \sigma)$. After the modified action a_k is sent to the process, the temporal difference Δ_k is calculated. If the actual performance is better than the predicted one, the actor is adapted toward the modified control action a_k.

Let the actor be represented by a neural network or a fuzzy system

$$a_k = a_k^\pi + \tilde{a}_k = \hat{\pi}(s_k, \varphi_k) + \tilde{a}_k, \tag{9.46}$$

where φ_k is a vector of adjustable parameters. To update φ_k, the following learning rule is used:

$$\varphi_{k+1} = \varphi_k + \alpha_a \Delta_k \tilde{a}_k \frac{\partial \hat{\pi}(s_k, \varphi_k)}{\partial \varphi_k}, \tag{9.47}$$

where $\alpha_a > 0$ is the actor's learning rate.

9.5. Exploration

9.5.1. Exploration vs. Exploitation

When the agent always chooses the action that corresponds to the highest action value or leads to a new state with the highest value, it will find a solution, but not necessarily the optimal solution. The reason for this is that some state-action combinations have never been evaluated, as they did not correspond to the highest value at that time. In RL, exploration is explained as the need of the agent to try out suboptimal actions in its environment, as for to guarantee that it learns the optimal policy instead of some suboptimal one. This sounds paradoxical; the need for choosing suboptimal actions, in order to learn an optimal policy. But then realize that exploration is not a concept solely for the RL framework. In real life, exploration is an every-day experience. For human beings, as well as most animals, exploration is our curiosity that drives us to know more about our own environment. This might have some negative immediate effects, such as pain and time delay, but whenever we explore, we hope to gain more long-term reward than we would have experienced without it (practice makes perfect!).

Example 9.2 Imagine you find yourself a new job and you have to move to a new town for it. Since you are not yet familiar with the road network, you consult an online route planner to find out the best way to drive to your office. For days and weeks, you follow this route, but at a moment you get annoyed by the large number of traffic lights that you encounter. For some reason, these traffic lights appear to always make you stop. You start wondering if there could be no faster way to drive. You decide to turn left in a street, but after some time, you realize that this street does not take you to your office. The next day, you decide to stick to your safe route, although it might take you a little longer. Then, at

one moment, when the traffic is highly annoying and you are already late for work, you decide to take another chance and take a street to the right. To your surprise, you find that although this road is a little longer, it is much more quiet and takes you to your office really fast. You have found yourself a better policy for driving to your office. From now on, you will take this route and sometimes explore to find perhaps even faster ways.

□

An explorative action thus might only appear to be suboptimal. At that time, the action was suboptimal, but when the policy converges to the optimal policy, it can prove to be optimal. Exploitation is using the current knowledge for taking actions. This is observed in real-life when we have practiced a particular task long enough to be satisfied with the resulting reward. Typically, the older a person gets (the longer he or she operates in his or her environment) the more he or she will exploit his or her knowledge rather than explore alternatives. Exploration is especially important when the agent acts in an environment in which the system dynamics change over time. At all times, the issue is whether to exploit already gained knowledge or to explore unknown regions of state-space. This issue is known as the *exploration vs. exploitation dilemma*. How an agent can exploit its current knowledge about the values functions and still explore a fraction of the time is the subject of this section.

9.5.2. Undirected Exploration

Undirected exploration is the most basic form of exploration. Random action selection is used to let the agent deviate from the current optimal policy in the hope of finding a policy that is closer to the real optimum. There are some different undirected exploration techniques that will now be described.

ϵ-greedy

In ϵ-greedy action selection, there is a probability of ϵ of selecting a random action instead of the one with the highest Q-value. This method is slow, since also actions that dot not appear promising will eventually be explored, but it is guaranteed to lead to the optimal policy in the long run. During the course of learning, the value of ϵ can be annealed (gradually decreased) to ensure that after a certain period of time there is no more exploration, but only greedy action selection.

Max-Boltzmann

With Max-Boltzmann, or *soft-max* exploration, there is also a possibility of ϵ to choose an action at random, but then according to a special probability function, known as the Boltzmann-Gibbs probability density function. The probability $P\{a \mid s\}$ of selecting an action a given state s and Q-values $Q(s, i)$ for all $i \in A$ is computed as follows:

$$P\{a \mid s\} = \frac{e^{Q(s,a)/\tau}}{\sum_i e^{Q(s,i)/\tau}}, \tag{9.48}$$

where τ is a 'temperature' variable, which is used for annealing the amount of exploration. High values for τ causes all actions to be nearly equiprobable, whereas low values for τ cause greater differences in the probabilities.

When the Q-values for different actions in one state are all nearly the same, the possibilities of choosing each action are also almost equal. This results in strong exploration. When the differences between these Q-values are larger, the amount of exploration will be lower, since the probability of choosing the greedy action is significantly higher than that of choosing an explorative action. This undirected exploration strategy corresponds to the intuition that one should explore when the current best option is not expected to be much better than the alternatives. Still, because exploration decreases when some actions are regarded to be much better than some other actions that were not yet tried, there is no guarantee that it leads to finding the optimal policy.

Optimistic Initial Values

Another (very simple) exploration strategy is to initialize the Q-function with high values, i.e. values at the upper bound of the optimal Q-function. When an action is taken, the corresponding Q-value will decrease to a value closer to the optimal value. The next time the agent is in that state, it will choose an action amongst the ones never taken, since these correspond to the highest Q-values. This strategy thus guarantees that all actions will be explored in every state. It results in initial high exploration. On the other hand, learning might take a very long time this way as all state-action pairs and thus many policies will be evaluated.

9.5.3. Directed Exploration *

In directed exploration, an *exploration reward function* is created that assigns rewards to trying out particular parts of the state space. The exploration is thus not completely random, as in undirected exploration. A higher level system determines in parallel to the learning and decision making, which parts of the state space should be explored. This section will discuss some of these higher level systems of directed exploration.

Frequency Based Exploration

Reward-based exploration means that a reward function, denoted as $R^E(s, a, s')$ (Wiering, 1999) assigns rewards for trying out different parts of the state space. Based on these rewards, the exploration that is chosen is expected to result in a maximum increase of experience.

For the frequency based approach, the action that has been executed least frequently will be selected for exploration. The reward function is then:

$$R^E(s, a, s_i) = -\frac{C_s(a)}{K_C}, \quad \forall i, \tag{9.49}$$

where $C_s(a)$ is the local counter, counting the number of times the action a is taken in state s. K_C is a scaling constant. The exploration rewards are treated in the same way as normal rewards are. They can also be discounted, so the resulted exploration behavior can be quite complex (Wiering, 1999).

Recency Based Exploration

In recency based exploration, the exploration reward is based on the time since that action was last executed. The reward function is:

$$R^E(s, a, s_i) = \frac{-k}{K_T}, \quad \forall i, \tag{9.50}$$

where k is the global time counter, indicating the discrete time at the current step. K_T is a scaling constant. According to Wiering (1999), this exploration reward rule is especially valuable when the agent interacts with a changing environment.

Error Based Exploration

The error based reward function chooses actions that lead to strongly increasing Q-values. The reward rule is defined as follows:

$$R^E(s_k, a, s_{k+1}) = Q_{k+1}(s_k, a_k) - Q_k(s_k, a_k) - K_P, \tag{9.51}$$

where $Q_k(s_k, a_k)$ denotes the Q-value of the state-action pair before the update and $Q_{k+1}(s_k, a_k)$ the one after the last update, which was computed before computing this exploration reward. The constant K_P ensures that the exploration reward is always negative (Wiering, 1999).

9.5.4. Dynamic Exploration *

A novel exploration method, proposed in (van Ast and Babuška, 2006), is called *dynamic exploration*. It is inspired by the observation that in real-world RL problems, long sequences of the same action must often be selected in order to reach the goal. Dynamic exploration selects with a higher probability the action that has been chosen in the previous time step. It has been shown that for typical gridworld search problems, this strategy speeds up the convergence of learning considerably. The general form of dynamic exploration is given by:

$$P(s_k, a_k) = f(s_k, a_{k-1}, a_{k-2}, \ldots), \tag{9.52}$$

where $P(s_k, a_k)$ is the probability of choosing action a in state s at the discrete time step k.

Example 9.3 We regard as an illustration the grid-search problem in Figure 9.5.

The agent's goal is to find the shortest path from the start cell S to the goal cell G. In every state, the agent can choose between the actions North, East, South and West. Free states are white and blocked states are grey. In free states, the agent receives a reward of -1, in the blocked states the reward is -5 (and stays in the previous state) and at reaching the goal, it receives $+10$. Note that for efficient exploration, the agent must take relatively long sequences of identical actions. We refer to these states as *long-path states*. Still, at some crucial states, called the *switch states*, the agent must select different actions.

□

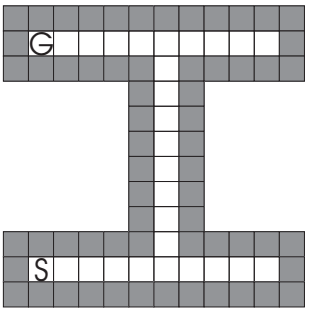


Figure 9.5.: Example of a grid-search problem with a high proportion of long-path states.

The states belong to long paths when the optimal policy selects the same action as in the state one time step before. The states belong to switch states when the optimal policy selects actions that are not the same as in the state one time step before. We will denote:

$\mathcal{S}$	Set of all states in the quantized state-space
$s_k \in \mathcal{S}$	One particular state at time step k
$\mathcal{S}_l \subset \mathcal{S}$	Set of long-path states
$\mathcal{S}_s \subset \mathcal{S}$	Set of switch states

It holds that $\mathcal{S}_l \cup \mathcal{S}_s = \mathcal{S}$ and $\mathcal{S}_l \cap \mathcal{S}_s = \emptyset$. Further, denote $|\mathcal{S}|$ the number of elements in $\mathcal{S}$. The basic idea of dynamic exploration is that for realistic, real-world problems $|\mathcal{S}_l| > |\mathcal{S}_s|$. If we define $p = \frac{|\mathcal{S}_l|}{|\mathcal{S}|}$, this means that $p > 0.5$. In many real-world applications of RL, the agent needs to select long sequences of the same action in order to reach the goal state. With exploration strategies where pure random action selection is used, the agent might waste a lot of time on parts of the state-space where long sequences of the same action are required. With dynamic exploration, the action selection is dynamic, i.e., depends on previously chosen actions. With a probability of ϵ, an explorative action is chosen according to the following probability distribution:

$$P(s_k, a_k) = f(s_k, a_{k-1}, a_{k-2}, \ldots), \tag{9.53}$$

where $P(s_k, a_k)$ is the probability of choosing action a in state s at the discrete time step k. A particular implementation of this function is:

$$P\{a_k \mid s_k, a_{k-1}\} = \begin{cases} (1-\beta)\,\Pi(s_k, a_k) + \beta & \text{if } a_k = a_{k-1} \\ (1-\beta)\,\Pi(s_k, a_k) & \text{if } a_k \neq a_{k-1} \end{cases}, \tag{9.54}$$

with β a weighting factor, $0 \leq \beta \leq 1$ and $\Pi(s_k, a_k)$ a soft-max policy:

$$\Pi(s_k, a_k) = \frac{e^{Q(s_k, a_k)}}{\sum_b^N e^{Q(s_k, b_k)}}, \tag{9.55}$$

where N denotes the total number of actions.

The parameter β can be thought of as an inertia. For instance, in a gridworld, the agent favors to continue its movement in the direction it is moving. Note that the general case of eq. (9.54) with $\beta = 0$ results in the max-Boltzmann exploration rule with a temperature parameter $\tau = 1$.

Example 9.4 In this example we use the RL method of Q-learning, which is discussed in Section 9.4.4. We evaluate the performance of dynamic exploration compared to max-Boltzmann exploration (where $\beta = 0$) for random gridworlds with 20% of the states blocked. Random gridworlds are more general than the illustrative gridworld from Figure 9.5. They are an abstraction of the various robot navigation tasks used in the literature, such as in (Thrun, 1992; Wiering, 1999; Bakker, 2002). Figure 9.6 shows an example of a random gridworld with 20% blocked states.

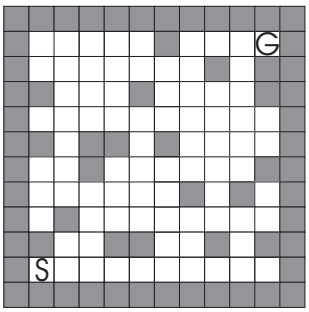
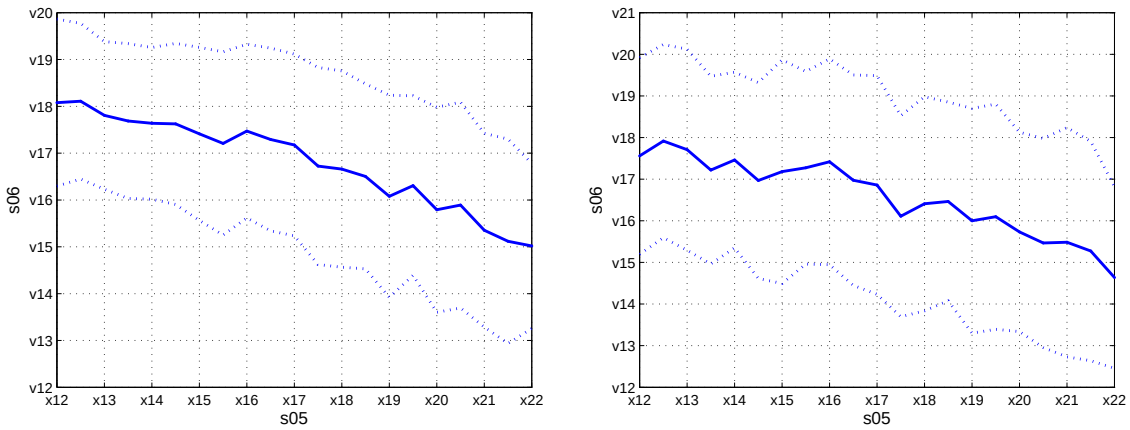


Figure 9.6.: Example of a 10×10 random gridworld with 20% blocked states.

We simulated two cases of gridworlds: with and without noise. A noisy environment is modeled by the transition probability, i.e. the probability that the agent moves according to its chosen action. The learning algorithm is plain tabular Q-learning with the rewards as specified above. The discounting factor is $\gamma = 0.95$ and the eligibility trace decay factor $\lambda = 0.5$. The learning rate is $\alpha = 1$. An explorative action is chosen with probability $\epsilon = 0.1$. The learning stops after 9 trials, as for this problem, the agent always converges to the optimal policy within this many trials. The results are presented in Figure 9.7.



(a) Transition probability = 1.

(b) Transition probability = 0.8.

Figure 9.7.: Learning time vs. β for 100 simulations.

It shows that the maximum improvement occurs for $\beta = 1$. The average maximal

improvements are 23.4% and 17.6% for the noiseless and the noise environment respectively. □

In the example and in (van Ast and Babuška, 2006) it has been shown that for grid search problems, $\beta = 1$, or full dynamic exploration is the optimal choice. It has been shown that a simple implementation of eq. (9.52) can considerably shorten the learning time compared to undirected exploration. Its main advantages compared to directed exploration (Wiering, 1999) are its straightforward implementation, lower complexity and applicability to Q-learning.

9.6. Applications

In the literature, many examples can be found in which RL was successfully applied. Most of these applications are test-benches for new analysis or algorithms, but some are also real-world problems. They can be divided into three categories:

- control problems
- grid-search problems
- games

In each of these categories, both test-problems and real-world problems are found. Test-problems are in general simplifications of real-world problems.

Robot control is especially popular for RL, but applications are also found for elevator dispatching (Sutton and Barto, 1998) and other problems in which planning is the central issue. For robot control, applications like swimming (Coulom, 2002) and the walking of a six legged machine (Kirchner, 1998) are found. More common problems are generally used as test-problem and include all kinds of variations of pendulum problems. Examples of these are the acrobot (Boone, 1997), the double pendulum (Randlov et al., 2000), single pendulum swing-up and balance (Perkins and Barto., 2001) and rotational pendulums, like the Furuta pendulum (Aamodt, 1997).

Mazes provide less exciting problems, but they are especially useful for RL, because one can often determine the optimal policy by inspecting the maze. Eventually, real-world maze problems can be found in adaptive routing on the internet, or other shortest path problems. In his Ph.D. thesis, Bakker (2004) uses all kinds of mazes for his RL algorithms. In (van Ast and Babuška, 2006) random mazes are used to demonstrate dynamic exploration.

RL has also been successfully applied to games. The most well known and exciting application is that of an agent that learned to play backgammon at a master level (Tesauro, 1994). Another problem, that is used as test-problem is Tic Tac Toe (Sutton and Barto, 1998).

The remainder of this section treats two applications that are typically illustrative to reinforcement learning for control. The first one is the pendulum swing-up and balance problem, where we will apply Q-learning. The second application is the inverted pendulum controlled by actor-critic RL.

9.6.1. Pendulum Swing-Up and Balance

The pendulum swing-up and balance problem is a challenging control problem for RL. In this problem, a pole is attached to a pivot point. At this pivot point, a motor can exert a torque to the pole. This torque causes the pole to rotate around the pivot point. However, the amount of torque is limited in such way, that it does not provide not enough force to move the pendulum to its upside-down position. In order to do so, it must gain energy by making the pendulum swing back and forth until it reaches the top. This is the first control task. The second control task is to keep the pendulum stable in its upside-down position.

The task is difficult for the RL agent for two reasons. The first is that it needs to perform an exact sequence of actions in order to swing-up and balance the pendulum. With a different sequence, it might never be able to complete the task. The second difficulty is that the agent can hardly value the effect of its actions before the goal is reached. This difficulty is known as *delayed reward*.

The pendulum problem is a simplification of more complex robot control problems. Its behavior can be easily investigated, while RL is still challenging.

The System Dynamics

The pendulum is modeled by the non-linear state equations in eq. (9.56).

$$J\dot{\omega} = Ku - mgR\sin(\theta) - b\omega, \tag{9.56}$$

with the angle (θ) and the angular velocity (ω) as the state variables and u the input. The constants J, K, g, R and b are not explained in detail. It is easy to verify that its equilibria are $(\theta, \omega) = (0, 0) \bigwedge (\pi, 0)$. By linearizing the non-linear state equations around its equilibria it can be found that the first equilibrium is stable, while the second one is unstable.

The pendulum is depicted in Figure 9.8(a). It also shows the conventions for the state variables.

Quantization of the State Variables

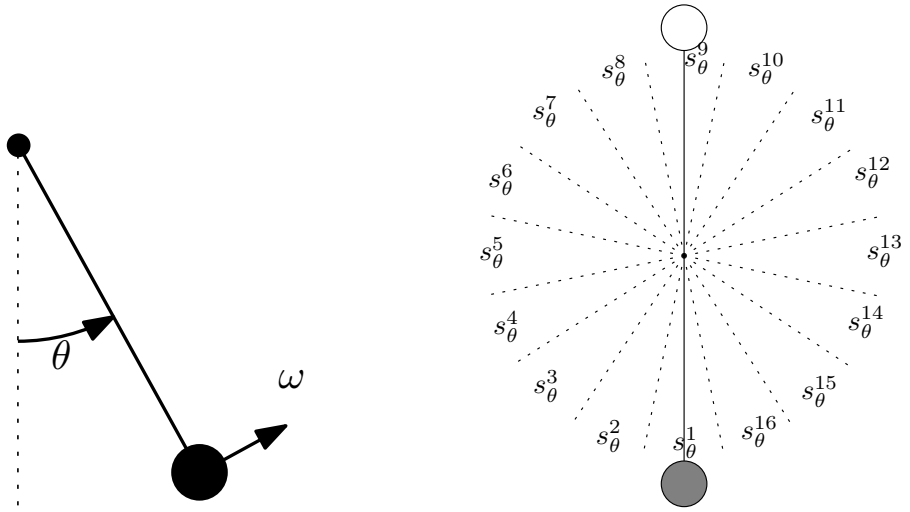
In order to use classical RL methods, the state variables needs to be quantized. Equal bin size will used to quantize the state variable θ. Figure 9.8(b) shows a particular quantization of the angle. The bins are positioned in such way that both equilibria fall in one bin and not at the boundary of two neighboring bins. The other state variable ω is quantized with boundaries from the set $\mathcal{S}_\omega$:

$$\mathcal{S}_\omega = \{-9, \ldots, 9\}[\text{ rad } s^{-1}], \tag{9.57}$$

where in place of the dots, the boundaries will be equally separated. The quantization is as follows:

$$s_\omega = \begin{cases} 1 & \text{for } \omega \leq \mathcal{S}_\omega^1, \\ i & \text{for } \mathcal{S}_\omega^i < \omega < \mathcal{S}_\omega^{i+1}, i = 2, 3, \ldots, N-1 \\ N & \text{for } \omega \geq \mathcal{S}_\omega^N, \end{cases} \tag{9.58}$$

where $\mathcal{S}_\omega^i$ is the i-th element of the set $\mathcal{S}_\omega$ that contains N elements.



(a) Convention for θ and ω.

(b) A quantization of the angle θ in 16 bins.

Figure 9.8.: The pendulum swing-up and balance problem.

The Action Set

The pendulum swing-up problem is a typical under-actuated problem, i.e. the torque applied to it is too small to directly swing-up the pendulum. The learning agent should therefore learn to swing the pendulum back and forth, thereby accumulating (potential and kinetic) energy. As most RL algorithms are designed for discrete-time control, a gap has to be bridged between the learning algorithm and the continuous-time dynamics of the pendulum. A regular way of doing this is to associate actions with applying torques. This can be either at a low-level, where each action is *directly* associated with a torque, or at a higher level, where an action is associated with a control law, which in turn determines the torque. Several different actions can be defined, which are shown in Table 9.1. The parameter μ is the maximum torque that can be applied.

a_1:	$+\mu$
a_2:	0
a_3:	$-\mu$
a_4:	sign $(\omega)\mu$
a_5:	$1 \cdot (\pi - \theta)$

Table 9.1.: Possible actions for the pendulum problem

A possible action set can be $A_{ll} = \{a_1, a_2, a_3\}$ (low-level action set) representing constant maximal torque in both directions, corresponding to bang-bang control, with the possibility to exert no torque at all. An other, higher level, action set can consist of the other two actions $A_{hl} = \{a_4, a_5\}$. In this set, a_4 is to accumulate energy to swing the pendulum in the direction of the unstable equilibrium and a_5 is to balance it with a proportional gain of 1. It can be proven that a_4 is the most efficient way to destabilize the pendulum. In this

problem we make sure that also the control output of A_{hl} is limited to $\pm\mu$.

It makes a great difference for the RL agent which action set it can use. As long sequences of positive or negative maximum torques are necessary to complete the task, it is very unlikely for an RL agent to find the correct sequence of low-level actions, especially with a fine state quantization. On the other hand, the higher level action set makes the problem somewhat easier, since the agent effectively only has to learn how to switch between the two actions in the set. Furthermore, the control laws make the system symmetrical in the angle.

In order to design the higher level action set, the dynamics of the system needed to be known in advance. The low-level action set is much more basic to design. Prior knowledge in the action set is thus expected to improve RL performance.

The Reward Function

The reward function is also a very important part of reinforcement learning. Since it essentially tells the agent what it should accomplish, the success of the learning strongly depends on it. The reward function determines in each state what reward it provides to the agent. For the pendulum, there are three kinds of states, namely:

- The goal state
- Four trap states
- The other states

We define a trap state as a state in which the torque is cancelled by the gravitational force, making the pendulum balance at a different angle than $\theta = \pi$, or $\theta = 0$. These angles can be easily derived from the non-linear state equation in eq. (9.56). A good reward function for a time optimal problem assigns positive or zero reward to the agent in the goal state and negative rewards in both the trap state and the other states. Intuitively, one would also reward the agent for getting in the trap state more negatively than for getting to one of the other states. A possible reward function like this can be:

- $R = +10$ in the goal state
- $R = -10$ in the trap state
- $R = -1$ anywhere else

Some prior knowledge could be incorporated in the design of the reward function. E.g. a reward function that assigns reward proportional to the potential energy of the pendulum would also make the agent favor to swing-up the pendulum. A reward function like this is:

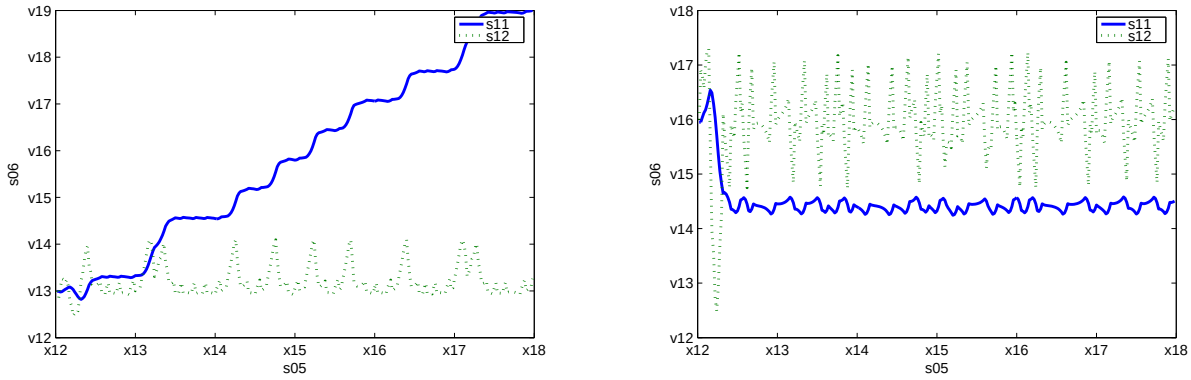
$$R(\theta) = -\cos(\theta), \tag{9.59}$$

where the reward is thus dependent on the angle. One should however always be very careful in shaping the reward function in this way. In (Sutton and Barto, 1998) the authors warn the designer of the RL task by saying that the reward function should only tell the agent *what* it should do and not *how*.

In the experiment, we use the low-level action set A_{ll}. The discount factor is chosen to be $\gamma = 0.95$ and the method of replacing traces is used with a trace decay factor of $\lambda = 0.9$. Furthermore, standard ϵ-greedy is the exploration strategy with a constant $\epsilon = 0.01$. The learning is stopped when there is no change in the policy in 10 subsequent trials.

Results

During the course of learning, the agent may find policies that are sub-optimal before learning the optimal one. Figure 9.9 shows the behavior of two policies, one in the middle of the learning and one from the end.

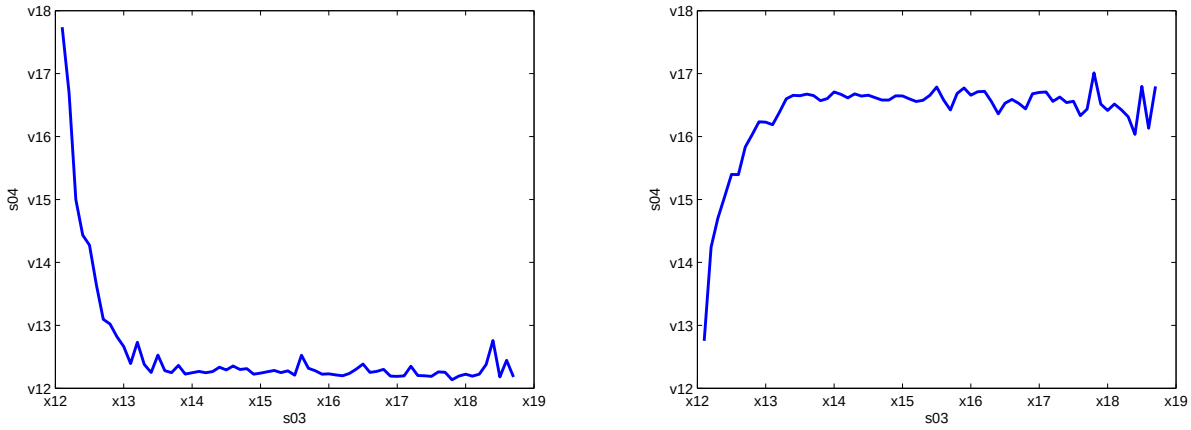


(a) Behavior in the middle of the learning.

(b) Behavior at the end of the learning.

Figure 9.9.: Behavior of the agent during the course of the learning.

The learning curves, showing the number of iterations and the accumulated reward per trial are shown in Figure 9.10. These experiments show that $Q(\lambda)$-learning is able to find a (near-)optimal policy for the pendulum control problem.



(a) Number of episodes per trial.

(b) Total reward per trial.

Figure 9.10.: Learning performance for the pendulum resulting in an optimal policy.

9.6.2. Inverted Pendulum

In this problem, reinforcement learning is used to learn a controller for the inverted pendulum, which is a well-known benchmark problem. The aim is to learn to balance the pendulum in its upright position by accelerating the cart left and right as depicted in Figure 9.11.

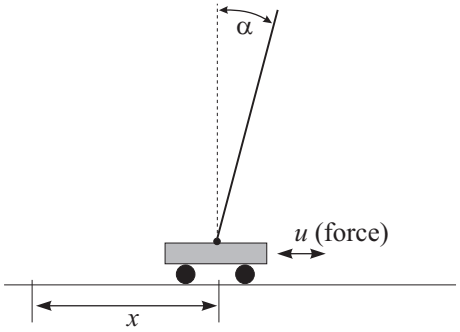


Figure 9.11.: The inverted pendulum.

The system has one input u, the acceleration of the cart, and two outputs, the position of the cart x and the pendulum angle α. When a mathematical or simulation model of the system is available, it is not difficult to design a controller. Figure 9.12 shows a block diagram of a cascaded PD controller that has been tuned by a trial and error procedure on a Simulink model of the system (`invpend.mdl`). Figure 9.13 shows a response of the PD controller to steps in the position reference.

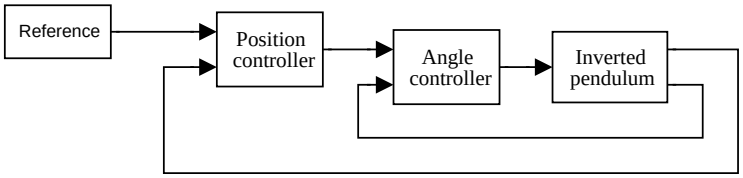


Figure 9.12.: Cascade PD control of the inverted pendulum.

For the RL experiment, the inner controller is made adaptive, while the PD position controller remains in place. The goal is to learn to stabilize the pendulum, given a completely void initial strategy (random actions).

The critic is represented by a singleton fuzzy model with two inputs, the current angle α_k and the current control signal u_k. Seven triangular membership functions are used for each input. The membership functions are fixed and the consequent parameters are adaptive. The initial value is -1 for each consequent parameter.

The controller is also represented by a singleton fuzzy model with two inputs, the current angle α_k and its derivative $\frac{\mathrm{d}\alpha}{\mathrm{d}t}_k$. Five triangular membership functions are used for each input. The membership functions are fixed and the consequent parameters are adaptive. The initial value is 0 for each consequent parameter. The initial control strategy is thus completely determined by the stochastic action modifier (it is thus random). This, of course, yields an unstable controller. After several control trials (the pendulum is reset to

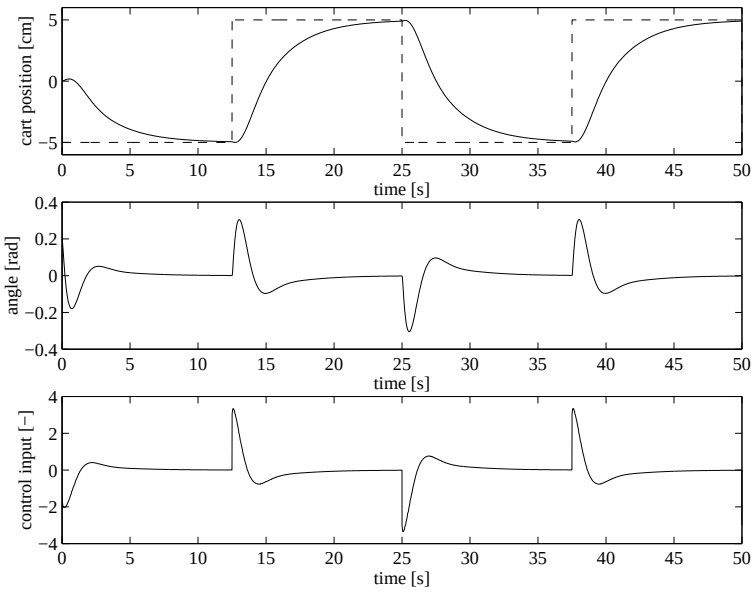


Figure 9.13.: Performance of the PD controller.

its vertical position after each failure), the RL scheme learns how to control the system (Figure 9.14).

Note that up to about 20 seconds, the controller is not able to stabilize the system. After about 20 to 30 failures, the performance rapidly improves and eventually it approaches the performance of the well-tuned PD controller (Figure 9.15). To produce this result, the final controller parameters were fixed and the noise was switched off.

Figure 9.16 shows the final surfaces of the critic and of the controller. Note that the critic highly rewards states when $\alpha = 0$ and $u = 0$. States where both α and u are negative are penalized, as they lead to failures (control action in the wrong direction). States where α is negative but u is positive (and vice versa) are evaluated in between the above two extremes. These control actions should lead to an improvement (control action in the right direction).

9.7. Conclusions

This chapter introduced the concept of reinforcement learning (RL). The basic principle of RL, where the agent interacts with an environment from which it only receives rewards have been discussed. The agent uses a value function to process these rewards in such a way that it improves its policy for solving the RL problem. This value function assigns values to states, or to state-action pairs. These values indicate what action should be taken in each state to solve the RL problem. When an explicit model of the environment is provided to the agent, the Bellman optimality equations can be solved by dynamic programming methods, such as value iteration and policy iteration. When this model is not available, methods of temporal difference have to be used. Several of these methods, like Q-learning, SARSA and the actor-critic architecture have been discussed. When acting in an unknown environment, it important for the agent to effectively explore. Different

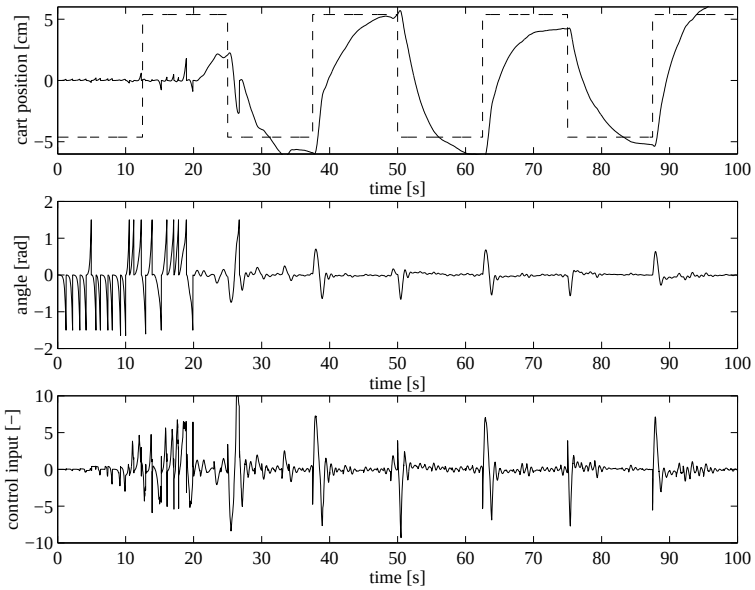


Figure 9.14.: The learning progress of the RL controller.

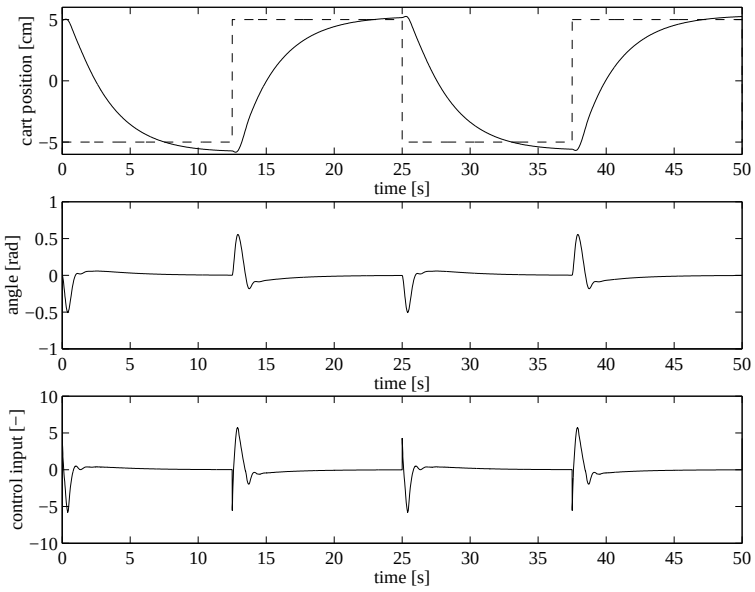
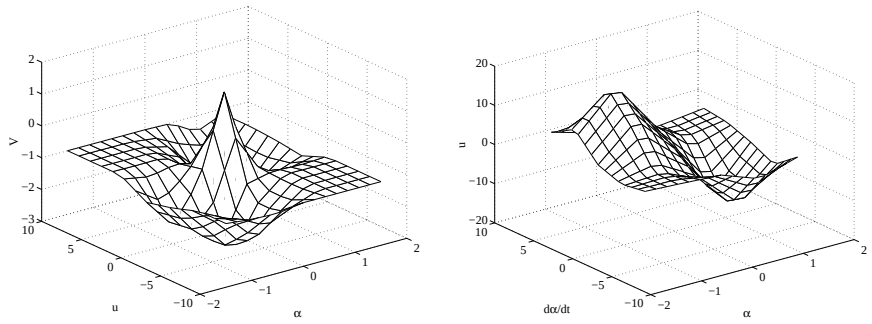


Figure 9.15.: The final performance the adapted RL controller (no exploration).



(a) Critic. (b) Controller.

Figure 9.16.: The final surface of the critic (left) and of the controller (right).

exploration methods, like undirected, directed and dynamic exploration have been discussed. Three problems have been discussed, namely the random gridworld, the pendulum swing-up and balance and the inverted pendulum. Other researchers have obtained good results from applying RL to many tasks, ranging from riding a bike to playing backgammon.

9.8. Problems

1. Prove mathematically that the discounted sum of immediate rewards

$$R_k = \sum_{n=0}^{\infty} \gamma^n r_{k+n+1}$$

   is indeed finite when $0 \leq \gamma < 1$. Assume that the agent always receives an immediate reward of -1. Compute R_k when $\gamma = 0.2$ and $\gamma = 0.95$. Compare your results and give a brief interpretation of this difference.

2. Explain the difference between the V-value function and the Q-value function and give an advantage of both.

3. Explain the difference between on-policy and off-policy learning.

4. Is actor-critic reinforcement learning an on-policy or off-policy method? Explain you answer.

5. Describe the tabular algorithm for $Q(\lambda)$-learning, using replacing traces to update the eligibility trace.

A. Ordinary Sets and Membership Functions

This appendix is a refresher on basic concepts of the theory of ordinary[1] (as opposed to fuzzy) sets. The basic notation and terminology will be introduced.

Definition A.1 (Set) *A set is a collection of objects with a certain property. The individual objects are referred to as elements or members of the set.*

Sets are denoted by upper-case letters and their elements by lower-case letters. The expression "x is an element of set A" is written as $x \in A$. The letter X denotes the universe of discourse (the universal set). This set contains all the possible elements in a particular context, from which sets can be formed. An important universal set is the Euclidean vector space $\mathbb{R}^n$ for some $n \in \mathbb{N}$. This is the space of all n-tuples of real numbers. There several ways to define a set:

- By specifying the properties satisfied by the members of the set:

  $$A = \{x \mid P(x)\},$$

  where the vertical bar $\mid$ means "such that" and $P(x)$ is a proposition which is true for all elements of A and false for remaining elements of the universal set X. As an example consider a set I of natural numbers greater than or equal to 2 and lower than 7: $I = \{x \mid x \in \mathbb{N}, 2 \leq x < 7\}$.

- By listing all its elements (only for finite sets):

  $$A = \{x_1, x_2, \ldots, x_n\}\,. \tag{A.1}$$

  The set I of natural numbers greater than or equal to 2 and less than 7 can be written as: $I = \{2, 3, 4, 5, 6\}$.

- By using a membership (characteristic, indicator) function, which equals one for the members of A and zero otherwise. As this definition is very useful in conventional set theory and essential in fuzzy set theory, we state it in the following definition.

Definition A.2 (Membership Function of an Ordinary Set) *The membership function of the set A in the universe X (denoted by $\mu_A(x)$) is a mapping from X to the set {0,1}: $\mu_A(x)\colon X \to \{0, 1\}$, such that:*

$$\mu_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A\,. \end{cases} \tag{A.2}$$

[1] Ordinary (nonfuzzy) sets are also referred to as *crisp sets*. In various contexts, the term crisp is used as an opposite to fuzzy.

The membership function is also called the *characteristic* function or the *indicator* function.

We will see later that operations on sets, like the intersection or union, can be conveniently defined by means of algebraic operations on the membership functions of these sets. Also in function approximation and modeling, membership functions are useful as shown in the following example.

Example A.1 (Local Regression) A common approach to the approximation of complex nonlinear functions is to write them as a concatenation of simpler functions f_i, valid locally in disjunct[2] sets A_i, $i = 1, 2, \ldots, n$:

$$y = \begin{cases} f_1(x), & \text{if } x \in A_1, \\ f_2(x), & \text{if } x \in A_2, \\ \vdots & \vdots \\ f_n(x), & \text{if } x \in A_n\,. \end{cases} \tag{A.3}$$

By using membership functions, this model can be written in a more compact form:

$$y = \sum_{i=1}^{n} \mu_{A_i}(x) f_i(x)\,. \tag{A.4}$$

Figure A.1 gives an example of a nonlinear function approximated by a concatenation of three local linear segments that valid local in subsets of X defined by their membership functions.

$$y = \sum_{i=1}^{3} \mu_{A_i}(x)(a_i x + b_i) \tag{A.5}$$

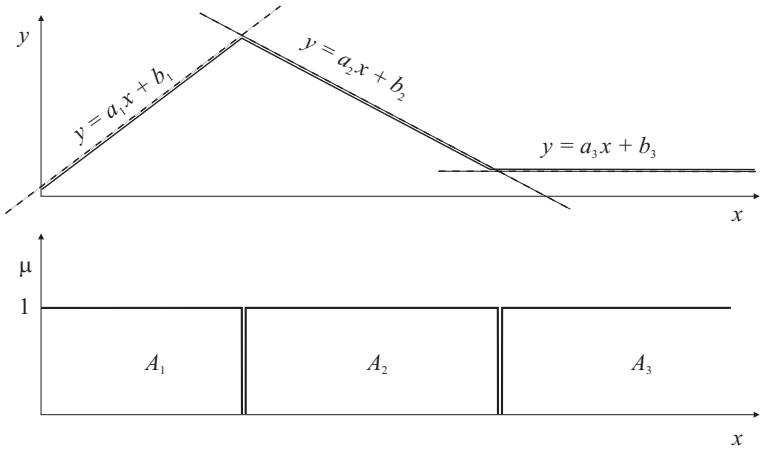


Figure A.1.: Example of a piece-wise linear function.

□

[2]Disjunct sets have an empty intersection.

The number of elements of a finite set A is called the *cardinality* of A and is denoted by $\mathrm{card}(A)$. A family of all subsets of a given set A is called the *power set* of A; denoted by $\mathcal{P}(A)$.

The basic operations of sets are the complement, the union and the intersection.

Definition A.3 (Complement) *The (absolute)* complement $\bar{A}$ *of* A *is the set of all members of the universal set* X *which are not members of* A*:*

$$\bar{A} = \{x \mid x \in X \text{ and } x \notin A\}.$$

Definition A.4 (Union) *The* union *of sets* A *and* B *is the set containing all elements that belong either to* A *or to* B *or to both* A *and* B*:*

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}.$$

The union operation can also be defined for a family of sets $\{A_i \mid i \in I\}$*:*

$$\bigcup_{i \in I} A_i = \{x \mid x \in A_i \text{ for some } i \in I\}.$$

Definition A.5 (Intersection) *The* intersection *of sets* A *and* B *is the set containing all elements that belong to both* A *and* B*:*

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}.$$

The intersection can also be defined for a family of sets $\{A_i \mid i \in I\}$*:*

$$\bigcap_{i \in I} A_i = \{x \mid x \in A_i \text{ for all } i \in I\}.$$

Table A.1 lists the result of the above set-theoretic operations in terms of membership degrees.

Table A.1.: Set-theoretic operations in classical set theory.

A	B	$A \cap B$	$A \cup B$	$\bar{A}$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Definition A.6 (Cartesian Product) *The* Cartesian product *of sets* A *and* B *is the set of all ordered pairs:*

$$A \times B = \{\langle a, b\rangle \mid a \in A, b \in B\}.$$

Note that if $A \neq B$ and $A \neq \emptyset$, $B \neq \emptyset$, then $A \times B \neq B \times A$. The Cartesian product of a family $\{A_1, A_2, \ldots, A_n\}$ is the set of all n-tuples $\langle a_1, a_2, \ldots, a_n\rangle$ such that $a_i \in A_i$ for every $i = 1, 2, \ldots, n$. It is written as $A_1 \times A_2 \times \cdots \times A_n$. Thus,

$$A_1 \times A_2 \times \cdots \times A_n = \{\langle a_1, a_2, \ldots, a_n\rangle \mid a_i \in A_i \text{ for every } i = 1, 2, \ldots, n\}.$$

The Cartesian products $A \times A$, $A \times A \times A$, etc., are denoted by A^2, A^3, etc., respectively. Subsets of Cartesian products are called relations.

B. MATLAB Code

The MATLAB code given in this appendix can be downloaded from the Web page (`http://www.dcsc.tudelft.nl/~sc42050`) or requested from the author at the following address:

Dr. Robert Babuška
Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
tel: +31 15 2785117, fax: +31 15 2786679
e-mail: `R.Babuska@tudelft.nl`
`http://www.dcsc.tudelft.nl/~rbabuska`

B.1. Fuzzy Set Class

A set of functions are available which define a new class "fset" (fuzzy set) under MATLAB and provide various methods for this class, including: display as a point-wise list, plot of the membership function (`plot`), intersection due to Zadeh (`min`), algebraic intersection (`*` or `prod`) and a number of other operations. For illustration, a few of these functions are listed below.

B.1.1. Fuzzy Set Class Constructor

Fuzzy sets are represented as structures with two fields (vectors): the domain elements (`dom`) and the corresponding membership degrees (`mu`):

```
function A = fset(dom,mu)
% constructor for a fuzzy set

if isa(mu,'fset'), A = mu; return; end;
A.mu = mu;
A.dom = dom;
A = class(A,'fset');
```

B.1.2. Set-Theoretic Operations

Set-theoretic operations are implemented as operations on the membership degree vectors, assuming, of course, that the domains are equal. Examples are the Zadeh's intersection:

```
function c = and(a,b)
% Intersection of fuzzy sets (min)

c = a; c.mu = min(a.mu,b.mu);
```

or the algebraic (probabilistic) intersection:

```
function c = mtimes(a,b)
% Algebraic intersection of fuzzy sets

c = a; c.mu = a.mu .* b.mu;
```

The reader is encouraged to implement other operators and functions and to compare the results on some sample fuzzy sets. Fuzzy sets can easily be defined using parametric membership functions (such as the trapezoidal one, `mftrap`) or any other analytic function, see `example1` and `example2`.

B.2. Gustafson–Kessel Clustering Algorithm

Follows a simple MATLAB function which implements the Gustafson–Kessel algorithm of Chapter 4. The FCM algorithm presented in the same chapter can be obtained by simply modifying the distance function to the Euclidean norm.

```
function [U,V,F] = gk(Z,c,m,tol)
% Clustering with fuzzy covariance matrix (Gustafson-Kessel algorithm)
%
% [U,V,F] = GK(Z,c,m,tol)
%-------------------------------------------------
% Input:  Z   ... N by n data matrix
%         c   ... number of clusters
%         m   ... fuzziness exponent (m > 1)
%         tol ... termination tolerance (tol > 0)
%-------------------------------------------------
% Output: U   ... fuzzy partition matrix
%         V   ... cluster means (centers)
%         F   ... cluster covariance matrices

%----------------- prepare matrices ---------------------------------
[N,n] = size(Z);                                % data size
N1 = ones(N,1); n1 = ones(n,1); c1 = ones(1,c);   % aux. variables
U = zeros(N,c);                                 % partition matrix
d = U;                                          % distance matrix
F = zeros(n,n,c);                               % covariance matrix
%----------------- initialize U -------------------------------------
minZ = c1'*min(Z); maxZ = c1'*max(Z);           % data limits
V = minZ + (maxZ - minZ).*rand(c,n);            % random centers
for j = 1 : c,
   ZV = Z - N1*V(j,:);
   d(:,j) = sum((ZV.^2)')';                     % distances
end;
d = (d+1e-100).^(-1/(m-1));                     % inverse dist.
U0 = (d ./ (sum(d')'*c1));                      % part. matrix
%----------------- iterate ------------------------------------------
while max(max(U0-U)) > tol                      % no convergence
  U = U0; Um = U.^m; sumU = sum(Um);            % aux. vars
  V = (Um'*Z)./(n1*sumU)';                      % clust. centers
  for j = 1 : c,                                % for all clusters
    ZV = Z - N1*V(j,:);                         % auxiliary var
    f = n1*Um(:,j)'.*ZV'*ZV/sumU(j);            % cov. matrix
    d(:,j)=sum(ZV*(det(f)^(1/n)*inv(f)).*ZV,2);   % distances
  end;
  d = (d+1e-100).^(-1/(m-1));                   % inverse dist.
  U0 = (d ./ (sum(d')'*c1));                    % part. matrix
end
%----------------- create final F and U -----------------------------
U = U0; Um = U.^m; sumU = n1*sum(Um);
for j = 1 : c,
  ZV = Z - N1*V(j,:);
  F(:,:,j) = n1*Um(:,j)'.*ZV'*ZV/sumU(1,j);
end;
%----------------- end of function ----------------------------------
```

C. Symbols and Abbreviations

Printing Conventions Lower case characters in bold print denote column vectors. For example, $\mathbf{x}$ and $\mathbf{a}$ are column vectors. A row vector is denoted by using the transpose operator, for example $\mathbf{x}^T$ and $\mathbf{a}^T$. Lower case characters in italic denote elements of vectors and scalars. Upper case bold characters denote matrices, for instance, $\mathbf{X}$ is a matrix. Upper case italic characters such as A denote crisp and fuzzy sets. Upper case calligraphic characters denote families (sets) of sets.

No distinction is made between variables and their values, hence x may denote a variable or its value, depending on the context. No distinction is made either between a function and its value, e.g., μ may denote both a membership function and its value (a membership degree). Superscripts are sometimes used to index variables rather than to denote a power or a derivative. Where confusion could arise, the upper index is enclosed in parentheses. For instance, in fuzzy clustering $\mu_{ik}^{(l)}$ denotes the ikth element of a fuzzy partition matrix, computed at the lth iteration. $(\mu_{ik}^{(l)})^m$ denotes the mth power of this element. A hat denotes an estimate (such as $\hat{y}$).

Mathematical symbols

A, B, ...	fuzzy sets
$\mathcal{A}$, $\mathcal{B}$, ...	families (sets) of fuzzy sets
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$	system matrices
$\mathbf{F}$	cluster covariance matrix
$\mathcal{F}(X)$	set of all fuzzy sets on X
$\mathbf{I}$	identity matrix of appropriate dimensions
K	number of rules in a rule base
M_{fc}	fuzzy partitioning space
M_{hc}	hard partitioning space
M_{pc}	possibilistic partitioning space
N	number of items (data samples, linguistic terms, etc.)
$\mathcal{P}(A)$	power set of A
$\mathcal{O}(\cdot)$	the order of
R	fuzzy relation
$\mathbb{R}$	set of real numbers
$\mathcal{R}_i$	ith rule in a rule base
$\mathbf{U} = [\mu_{ik}]$	fuzzy partition matrix
$\mathbf{V}$	matrix containing cluster prototypes (means)
$\mathbf{X}$	matrix containing input data (regressors)
X, Y	domains (universes) of variables x and y
$\mathbf{Z}$	data (feature) matrix
$\mathbf{a}$, b	consequent parameters in a TS model

c	number of clusters
$d(\cdot,\cdot)$	distance measure
m	weighting exponent (determines fuzziness of the partition)
n	dimension of the vector $[\mathbf{x}^T, y]$
p	dimension of $\mathbf{x}$
$u(k)$, $y(k)$	input and output of a dynamic system at time k
$\mathbf{v}$	cluster prototype (center)
$\mathbf{x}(k)$	state of a dynamic system
$\mathbf{x}$	regression vector
y	output (regressand)
$\mathbf{y}$	vector containing output data (regressands)
$\mathbf{z}$	data vector
β	degree of fulfillment of a rule
$\boldsymbol{\phi}$	eigenvector of $\mathbf{F}$
γ	normalized degree of fulfillment
λ	eigenvalue of $\mathbf{F}$
μ, $\mu(\cdot)$	membership degree, membership function
$\mu_{i,k}$	membership of data vector $\mathbf{z}_k$ into cluster i
τ	time constant
$\mathbf{0}$	matrix of appropriate dimensions with all entries equal to zero
$\mathbf{1}$	matrix of appropriate dimensions with all entries equal to one

Operators:

$\cap$	(fuzzy) set intersection (conjunction)
$\cup$	(fuzzy) set union (disjunction)
$\wedge$	intersection, logical AND, minimum
$\vee$	union, logical OR, maximum
$\mathbf{X}^T$	transpose of matrix $\mathbf{X}$
$\bar{A}$	complement (negation) of A
∂	partial derivative
$\circ$	sup-t (max-min) composition
$\langle \mathbf{x}, \mathbf{y} \rangle$	inner product of $\mathbf{x}$ and $\mathbf{y}$
$\mathrm{card}(A)$	cardinality of (fuzzy) set A
$\mathrm{cog}(A)$	center of gravity defuzzification of fuzzy set A
$\mathrm{core}(A)$	core of fuzzy set A
det	determinant of a matrix
diag	diagonal matrix
$\mathrm{ext}(A)$	cylindrical extension of A
$\mathrm{hgt}(A)$	height of fuzzy set A
$\mathrm{mom}(A)$	mean of maxima defuzzification of fuzzy set A
$\mathrm{norm}(A)$	normalization of fuzzy set A
$\mathrm{proj}(A)$	point-wise projection of A
$\mathrm{rank}(\mathbf{X})$	rank of matrix $\mathbf{X}$
$\mathrm{supp}(A)$	support of fuzzy set A

Abbreviations

ANN	artificial neural network
B&B	branch-and-bound technique
BP	backpropagation
COG	center of gravity
FCM	fuzzy c-means
FLOP	floating point operations
GK	Gustafson–Kessel algorithm
MBPC	model-based predictive control
MIMO	multiple–input, multiple–output
MISO	multiple–input, single–output
MNN	multi-layer neural network
MOM	mean of maxima
(N)ARX	(nonlinear) autoregressive with exogenous input
P(ID)	proportional (integral derivative controller)
RBF(N)	radial basis function (network)
RL	reinforcement learning
SISO	single–input, single–output
SQP	sequential quadratic programming

References

Aamodt, T. (1997). Intelligent control via reinforcement learning: State transfer and stabilization of a rotational pendulum. Bachelors Thesis, University of Toronto.

Anderson, C. W. (1987). Strategy learning with multilayer connectionist representations. In *Proceedings of the 4th International Workshop on Machine Learning*, pages 103–114, Irvine, USA.

Babuška, R. (1998). *Fuzzy Modeling for Control.* Kluwer Academic Publishers, Boston, USA.

Babuška, R. and Verbruggen, H. B. (1997). Constructing fuzzy models by product space clustering. In Hellendoorn, H. and Driankov, D., editors, *Fuzzy Model Identification: Selected Approaches*, pages 53–90. Springer, Berlin, Germany.

Babuška, R., Verbruggen, H. B., and van Can, H. J. L. (1999). Fuzzy modeling of enzymatic penicillin–G conversion. *Engineering Applications of Artificial Intelligence*, 12(1):79–92.

Bakker, B. (2002). Reinforcement learning with Long Short-Term Memory. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Proceedings of the Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

Bakker, P. B. (2004). *The State of Mind; Reinforcement Learning with Recurrent Neural Networks.* PhD thesis, University of Leiden / IDSIA.

Barron, A. R. (1993). Universal approximation bounds for superposition of a sigmoidal function. *IEEE Transactions on Information Theory*, 39:930–945.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuron like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):834–846.

Bezdek, J. C. (1980). A convergence theorem for the fuzzy ISODATA clustering algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):1–8.

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function.* Plenum Press, New York.

Bezdek, J. C., Coray, C., Gunderson, R., and Watson, J. (1981). Detection and characterization of cluster substructure, I. Linear structure: Fuzzy c-lines. *SIAM Journal on Applied Mathematics*, 40(2):339–357.

Bezdek, J. C. and Pal, S. K., editors (1992). *Fuzzy Models for Pattern Recognition.* IEEE Press, New York.

Bonissone, P. P. (1994). Fuzzy logic controllers: an industrial reality. In Zurada, J. M., II, R. J. M., and Robinson, C. J., editors, *Computational Intelligence: Imitating Life*, pages 316–327. IEEE Press, Piscataway, NJ.

Boone, G. (1997). Efficient reinforcement learning: Model-based acrobot control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 229–234.

Botto, M. A., van den Boom, T., Krijgsman, A., and Sá da Costa, J. (1996). Constrained nonlinear predictive control based on input-output linearization using a neural network. In *Proceedings of the 13th IFAC World Congress*, San Fransisco (CA), USA.

Brown, M. and Harris, C. (1994). *Neurofuzzy Adaptive Modelling and Control*. Prentice Hall, New York.

Chen, S. and Billings, S. A. (1989). Representation of nonlinear systems: The NARMAX model. *International Journal of Control*, 49:1013–1032.

Clarke, D. W., Mohtadi, C., and Tuffs, P. S. (1987). Generalised predictive control. Part 1: The basic algorithm. Part 2: Extensions and interpretations. *Automatica*, 23(2):137–160.

Coulom, R. (2002). *Reinforcement Learning Using Neural Networks, with Applications to MotorControl*. PhD thesis, Institute National Polytechnique de Grenoble.

Cybenko, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Driankov, D., Hellendoorn, H., and Reinfrank, M. (1993). *An Introduction to Fuzzy Control*. Springer, Berlin.

Dubois, D., Prade, H., and Yager, R. R., editors (1993). *Readings in Fuzzy Sets for Intelligent Systems*. Morgan Kaufmann Publishers. ISBN 1-55860-257-7.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York.

Dunn, J. C. (1974). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57.

Economou, C. G., Morari, M., and Palsson, B. O. (1986). Internal model control. 5. Extension to nonlinear systems. *Industrial & Engineering Chemistry Process Design and Development*, 25(2):403–411.

Filev, D. P. (1996). Model based fuzzy control. In *Proceedings of the 4th European Congress on Intelligent Techniques and Soft Computing (EUFIT)*, Aachen, Germany.

Fischer, M. and Isermann, R. (1998). Inverse fuzzy process models for robust hybrid control. In Driankov, D. and Palm, R., editors, *Advances in Fuzzy Control*, pages 103–127. Springer, Heidelberg, Germany.

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

Froese, T. (1993). Applying of fuzzy control and neuronal networks to modern process control systems. In *Proceedings of the European Congress on Intelligent Techniques and Soft Computing (EUFIT)*, volume II, pages 559–568, Aachen.

Gath, I. and Geva, A. B. (1989). Unsupervised optimal fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7:773–781.

Gustafson, D. E. and Kessel, W. C. (1979). Fuzzy clustering with a fuzzy covariance matrix. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pages 761–766, San Diego, CA, USA.

Harris, C. J., Moore, C., and Brown, M. (1993). *Intelligent Control, Aspects of Fuzzy Logic and Neural Nets*. World Scientific, Singapore.

Hathaway, R. J. and Bezdek, J. C. (1993). Switching regression models and fuzzy clustering. *IEEE Transactions on Fuzzy Systems*, 1(3):195–204.

Haykin, S. (1994). *Neural Networks*. Macmillan Maxwell International, New York.

Hellendoorn, H. (1993). Design and development of fuzzy systems at siemens r&d. In *Proceedings of the 2nd IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1365–1370, San Fransisco (CA), USA. IEEE.

Hirota, K., editor (1993). *Industrial Applications of Fuzzy Technology*. Springer, Tokyo.

Holmblad, L. P. and Østergaard, J.-J. (1982). Control of a cement kiln by fuzzy logic. In Gupta, M. and Sanchez, E., editors, *Fuzzy Information and Decision Processes*, pages 389–399. North-Holland.

Ikoma, N. and Hirota, K. (1993). Nonlinear autoregressive model based on fuzzy relation. *Information Sciences*, 71:131–144.

Jager, R. (1995). *Fuzzy Logic in Control*. PhD dissertation, Delft University of Technology, Delft, The Netherlands.

Jager, R., Verbruggen, H. B., and Bruijn, P. M. (1992). The role of defuzzification methods in the application of fuzzy control. In *Proceedings of the IFAC Symposium on Intelligent Components and Instruments for Control Applications*, pages 111–116, Málaga, Spain.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs.

James, W. (1890). *Psychology (Briefer Course)*. Holt, New York.

Jang, J.-S. R. (1993). ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Transactions on Systems, Man & Cybernetics*, 23(3):665–685.

Jang, J.-S. R. and Sun, C.-T. (1993). Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, 4(1):156–159.

Jang, J.-S. R., Sun, C.-T., and Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing; a Computational Approach to Learning and Machine Intelligence*. Prentice-Hall, Upper Saddle River.

Kaelbing, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kaymak, U. and Babuška, R. (1995). Compatible cluster merging for fuzzy modeling. In *Proceedings of the 4th IEEE International Conference on Fuzzy Systems and 2nd International Fuzzy Engineering Symposium.,*, pages 897–904, Yokohama, Japan.

Kirchner, F. (1998). Q-learning of complex behaviors on a six-legged walking machine. In *Proceedings of the NIPS Workshop on Abstraction and Hierarchy in Reinforcement Learning*.

Klir, G. J. and Folger, T. A. (1988). *Fuzzy Sets, Uncertainty and Information*. Prentice-Hall, New Jersey.

Klir, G. J. and Yuan, B. (1995). *Fuzzy sets and fuzzy logic; theory and applications*. Prentice Hall.

Krishnapuram, R. and Freg, C.-P. (1992). Fitting an unknown number of lines and planes to image data through compatible cluster merging. *Pattern Recognition*, 25(4):385–400.

Krishnapuram, R. and Keller, J. M. (1993). A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110.

Kuipers, B. and Aström, K. (1994). The composition and validation of heterogeneous control laws. *Automatica*, 30(2):233–249.

Lakoff, G. (1973). Hedges: a study in meaning criteria and the logic of fuzzy concepts. *Journal of Philosofical Logic*, 2:458–508.

Lawler, E. L. and Wood, E. D. (1966). Branch-and-bound methods: A survey. *Journal of Operations Research*, 14:699–719.

Lee, C. C. (1990a). Fuzzy logic in control systems: fuzzy logic controller - part I. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418.

Lee, C. C. (1990b). Fuzzy logic in control systems: fuzzy logic controller - part II. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):419–435.

Leonaritis, I. J. and Billings, S. A. (1985). Input-output parametric models for non-linear systems. *International Journal of Control*, 41:303–344.

Lindskog, P. and Ljung, L. (1994). Tools for semiphysical modeling. In *Proceedings of the 10th IFAC Symposium on System Identification*, volume 3, pages 237–242.

Ljung, L. (1987). *System Identification, Theory for the User*. Prentice-Hall, New Jersey.

Mamdani, E. H. (1974). Application of fuzzy algorithm for control of simple dynamic plant. In *Proceedings of the Institution of Electrical Engineers*, volume 121, pages 1585–1588.

Mamdani, E. H. (1977). Application of fuzzy logic to approximate reasoning using linguistic systems. *Fuzzy Sets and Systems*, 26:1182–1191.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Mutha, R. K., Cluett, W. R., and Penlidis, A. (1997). Nonlinear model-based predictive control of control nonaffine systems. *Automatica*, 33(5):907–913.

Nakamori, Y. and Ryoke, M. (1994). Identification of fuzzy prediction models through hyper-ellipsoidal clustering. *IEEE Transactions on Systems, Man and Cybernetics*, 24(8):1153–73.

Novák, V. (1989). *Fuzzy Sets and their Applications*. Adam Hilger, Bristol.

Novák, V. (1996). A horizon shifting model of linguistic hedges for approximate reasoning. In *Proceedings of the 5th IEEE International Conference on Fuzzy Systems*, pages 423–427, New Orleans, USA.

Oliveira, S. d., Nevistić, V., and Morari, M. (1995). Control of nonlinear systems subject to input constraints. In *Proceedings of the IFAC Nonlinear Control Design Symposium (NOLCOS)*, volume 1, pages 15–20, Tahoe City, California.

Onnen, C., Babuška, R., Kaymak, U., Sousa, J. M., Verbruggen, H. B., and Isermann, R. (1997). Genetic algorithms for optimization in predictive control. *Control Engineering Practice*, 5(10):1363–1372.

Pal, N. R. and Bezdek, J. C. (1995). On cluster validity for the fuzzy c-means model. *IEEE Transactions on Fuzzy Systems*, 3(3):370–379.

Passino, K. M. and Yurkovich, S. (1998). *Fuzzy Control*. Addison-Wesley, Massachusetts, USA.

Pedrycz, W. (1984). An identification algorithm in fuzzy relational systems. *Fuzzy Sets and Systems*, 13:153–167.

Pedrycz, W. (1985). Applications of fuzzy relational equations for methods of reasoning in presence of fuzzy data. *Fuzzy Sets and Systems*, 16:163–175.

Pedrycz, W. (1993). *Fuzzy Control and Fuzzy Systems (second, extended, edition)*. John Wiley and Sons, New York.

Pedrycz, W. (1995). *Fuzzy Sets Engineering*. CRC Press, Boca Raton, Fl.

Perkins, T. J. and Barto., A. G. (2001). Lyapunov-constrained action sets for reinforcement learning. In Brodley, C. and Danyluk, A., editors, *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 409–416. Morgan Kaufmann.

Peterson, T., Arkun, E. H. Y., and Schork, F. J. (1992). A nonlinear SMC algorithm and its application to a semibatch polymerization reactor. *Chemical Engineering Science*, 47(4):737–753.

Psichogios, D. C. and Ungar, L. H. (1992). A hybrid neural network – first principles approach to process modeling. *AIChE Journal*, 38:1499–1511.

Randlov, J., Barto, A. G., and Rosenstein, M. (2000). Combining reinforcement learning with a local control algorithm. In *Proceedings of the 7th International Conference on Machine Learning (ICML)*, pages 775–782, San Fransisco (CA), USA. Morgan Kaufmann.

Roubos, J. A., Mollov, S., Babuška, R., and Verbruggen, H. B. (1999). Fuzzy model based predictive control by using Takagi-Sugeno fuzzy models. *International Journal of Approximate Reasoning*, 22(1/2):3–30.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. and McClelland, J., editors, *Parallel Distributed Processing*. MIT Press, Cambridge, MA.

Ruspini, E. (1970). Numerical methods for fuzzy clustering. *Information Sciences*, 2(3):319–350.

Santhanam, S. and Langari, R. (1994). Supervisory fuzzy adaptive control of a binary distillation column. In *Proceedings of the 3rd IEEE International Conference on Fuzzy Systems*, volume 2, pages 1063–1068, Orlando, Fl.

Sousa, J. M., Babuška, R., and Verbruggen, H. B. (1997). Fuzzy predictive control applied to an air-conditioning system. *Control Engineering Practice*, 5(10):1395–1406.

Sugeno, M. (1977). Fuzzy measures and fuzzy integrals: a survey. In Gupta, M., Sardis, G., and Gaines, B., editors, *Fuzzy Automata and Decision Processes*, pages 89–102. North-Holland Publishers, Amsterdam, The Netherlands.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its application to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics*, 15(1):116–132.

Tanaka, K., Ikeda, T., and Wang, H. O. (1996). Robust stabilization of a class of uncertain nonlinear systems via fuzzy control: Quadratic stability, H_∞ control theory and linear matrix inequalities. *IEEE Transactions on Fuzzy Systems*, 4(1):1–13.

Tanaka, K. and Sugeno, M. (1992). Stability analysis and design of fuzzy control systems. *Fuzzy Sets and Systems*, 45(2):135–156.

Tani, T., Utashiro, M., Umano, M., and Tanaka, K. (1994). Application of practical fuzzy–PID hybrid control system to petrochemical plant. In *Proceedings of the 3d IEEE International Conference on Fuzzy Systems*, volume 2, pages 1211–1216.

Terano, T., Asai, K., and Sugeno, M. (1994). *Applied Fuzzy Systems*. Academic Press, Inc., Boston.

Tesauro, G. J. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219.

Thompson, M. L. and Kramer, M. A. (1994). Modeling chemical processes using prior knowledge and neural networks. *AIChE Journal*, 40:1328–1340.

Thrun, S. (1992). Efficient exploration in reinforcement learning. Cmu-cs-92-102, Carnegie Mellon University, Computer Science Department, Pittsburgh.

van Ast, J. M. and Babuška, R. (2006). Dynamic exploration in Q(λ)-learning. In *Proceedings of the International Joint Conference on Neural Networks*, pages 41–46. Delft University of Technology, Delft Center for Systems and Control, IEEE.

Voisin, A., Rondeau, L., Ruelas, R., Dubois, G., and Lamotte, M. (1995). Conditions to establish and equivalence between a fuzzy relational model and a linear model. In *Proceedings of the 3rd European Congress on Intelligent Techniques and Soft Computing (EUFIT)*, pages 523–528, Aachen, Germany.

Wang, L.-X. (1992). Fuzzy systems are universal approximators. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 1163–1170, San Diego, USA.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3/4):279–292.

Werbos, P. J. (1974). *Beyond regression: new tools for prediction and analysis in the behavior sciences*. PhD Thesis, Harvard University, Committee on Applied Mathematics.

Wiering, M. A. (1999). *Explorations in Efficient Reinforcement Learning*. PhD thesis, University of Amsterdam / IDSIA.

Xie, X. L. and Beni, G. A. (1991). Validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(8):841–846.

Yasunobu, S. and Miyamoto, S. (1985). Automatic train operation system by predictive fuzzy control. In Sugeno, M., editor, *Industrial Applications of Fuzzy Control*, pages 1–18. North-Holland.

Yi, S. Y. and Chung, M. J. (1993). Identification of fuzzy relational model and its application to control. *Fuzzy Sets and Systems*, 59:25–33.

Yoshinari, Y., Pedrycz, W., and Hirota, K. (1993). Construction of fuzzy models through clustering techniques. *Fuzzy Sets and Systems*, 54:157–165.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, 8:338–353.

Zadeh, L. A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics*, 1:28–44.

Zadeh, L. A. (1975). Calculus of fuzzy restrictions. In Zadeh, L., Fu, K.-S., Tanaka, K., and Shimura, M., editors, *Fuzzy Sets and Their Applications to Cognitive and Decision Processes*, pages 1–39. Academic Press, New York, USA.

Zhao, J. (1995). *Fuzzy logic in modeling and control.* PhD dissertation, CESAME, Louvain la Neuve, Belgium.

Zimmermann, H.-J. (1987). *Fuzzy Sets, Decision Making and Expert Systems.* Kluwer Academic Publishers, Boston.

Zimmermann, H.-J. (1996). *Fuzzy Set Theory and its Application.* Kluwer Academic Publishers, Boston, third edition.

Zurada, J. M., Marks II, R. J., and Robinson, C. J., editors (1994). *Computational Intelligence: Imitating Life.* IEEE Press, Piscataway, NJ.

Index