Introduction to fuzzy control systems

Article · April 2001

2 authors:

Guanrong Chen
City University of Hong Kong

Young Hoon Joo
Kunsan National University

Introduction to Fuzzy Control Systems

Guanrong Chen

Department of Electrical and Computer Engineering
University of Houston, Houston, Texas 77204-4793, USA
Tel: 713-743-4446 Fax: 713-743-4444 Email: gchen@uh.edu

Young Hoon Joo

Department of Control and Instrumentation Engineering
Kunsan National University, Kunsan 573-701, Chonbuk, Korea
Tel: 82-654-469-4706 Fax: 82-654-466-2086 Email: yhjoo@ks.kunsan.ac.kr

Abstract

In this article, the basic notion of fuzzy control systems is introduced. Motivation for using fuzzy systems in control applications is first given, followed by a description of the basic structure of a typical fuzzy system: its fuzzification, rule base establishment, and defuzzificcation components. Three main fuzzy control approaches, namely, a model-free and a model-based approach as well as a popular adaptive control approach are then introduced. Finally, as an essential subject in fuzzy control systems, fuzzy system modeling is discussed, including both system structure and system parameters identifications. For parameter identification, the classical least-squares method and the efficient genetic algorithm technique are discussed in detail.

Keywords:
adaptive control, fuzzy control, fuzzy system, genetic algorithm, least-squares, modeling

March 8, 2001

Contents

1 Introduction

Fuzzy control systems are developed based on fuzzy mathematics. And fuzzy mathematics is a branch of applied mathematics, which has found broad applications in many fields including statistics and numerical analysis, systems and control engineering, pattern recognition, signal and image processing, and biomedical engineering alike.

While talking about fuzzy mathematics and its applications, particularly in intelligent control systems, a typical question of common concern is first to be addressed: why is fuzzy mathematics necessary when there already exist well-developed deterministic as well as stochastic mathematics? An answer to this question will help understand similar questions such as why fuzzy systems theory is needed for intelligent control when classical and modern control theories have been successfully developed for about a century.

Indeed, deterministic mathematics has been developed for a long, long time. Historical record shows that the ancient Sumerians had a deterministic number system using sixty as the base, and had used a calendar around 3,500 B.C. This number system was passed on to the Babylonians, who created mathematical tables for simple computing, as early as 2,000 B.C. Ever since the ancient theories of numbers (integers) developed by the Chinese and Greeks, the deterministic mathematics has gone through a tremendously long way to become mature, which, as it stands today, is fairly complete, rigorous, and abstract. As a result, it has a very broad range of covering, and is literally very powerful.

Yet, mathematics found itself to be insufficient to handle many real-world problems, if it remains to be only deterministic. The ancient Chinese and Greeks had the concept of "random numbers," dated back to some years B.C. According to the mathematical theory of probability and statistics, historical record shows that the renowned French mathematician Fermat already systematically investigated the gambling problem raised by the Italian mathematicians Cardano and Tartaglia about some 200 years ago. In only about two centuries, stochastic mathematics has been fully developed with undoubted success in some other branches of mathematics and many real-world applications.

The history has witnessed that the development of mathematics never stopped. Because both deterministic and stochastic mathematics require precise knowledge and perfect information, such as certainty (crisp numbers, explicit functions, exact distributions, accurate means and variances, *etc.*), when data information is not ideal (only partial, vague, or even conflict) they cannot be applied to formulate and solve any problems. Ironically, the majority of real-world problems are very often inexactly formulated and imperfectly described. Therefore, there is an extremely strong demand from the modern technology and industry for new mathematics that can handle such abnormal and irregular problems. Fuzzy mathematics, beginning with the fundamental concept of fuzzy sets, was born under this urge in the mid 1960s, which was mainly attributed to Lotfi A. Zadeh [19].

Fuzzy set theory basically extends the classical set theory that has membership of its elements described by the classical characteristic function (either "is" or "is not" a member of the set), to allow for partial membership described by a membership function

(both "is" and "is not" a member of the set at the same time, with a certain degree of belongings to the set). More details about fuzzy set theory will be introduced in the next section of this article. Roughly, fuzzy membership functions bridge the idealized, crisp, sharp, and discontinuous gap between "yes" and "no," between "in" and "out," between "true" and "false," *etc.* As a result, it demonstrates great capabilities and flexibilities in solving many real-world problems which classical mathematics fails to handle.

Fuzzy mathematics was applied to control systems, in both theory and engineering, almost immediately after its birth. Advances in modern computer technology have been steadily backing up the fuzzy mathematics for coping with engineering systems of a broad spectrum, including many control systems that are too complex or too imprecise to tackle by conventional control theories and techniques. The essence of systems control is to achieve automation. For this purpose, a combination of fuzzy control technology and advanced computer facility available in the industry provides a promising approach that can mimic human thinking and linguistic control ability, so as to equip the control systems with certain degree of artificial intelligence. It has now been realized that fuzzy control systems theory and methods offer a simple, realistic, and successful alternative for the control of complex, imperfectly modeled, and highly uncertain engineering systems. Fuzzy control technology appears to have a bright future in many real-world applications; its great potential in industrial automation should be further explored.

2 Fuzzy Control Systems

In a sense, fuzzy systems can be "trained" and can "learn" how to perform throughout a control task, and they are considered as a kind of intelligent control systems. Basically, fuzzy controllers can incorporates some knowledge of human experts in a form of logical inference rules. These controllers can then act in a humanlike fashion, for example, in making "decisions" as to what actions to take under various conditions.

The main signature of fuzzy logic technology is its ability of suggesting an approximate solution to an imprecisely formulated problem, which classical (two-valued) logic cannot offer. From this point of view, fuzzy logic is closer to human reasoning than the classical logic, where the latter attempts to precisely formulate and exactly solve a problem, in a way consistent with the classical, deterministic mathematics.

2.1 A Motive for Fuzzy Control Systems Theory

Conventional control systems theory, developed based on classical mathematics and the two-valued logic, is relatively complete, especially for linear dynamical systems. This theory has its solid foundation built on contemporary mathematical science, electrical engineering, and computer technology. It can provide very rigorous analysis and often perfect solutions when a system is precisely defined in terms of classical mathematics. Within this framework, some relatively advanced control techniques such as adaptive, robust, and nonlinear control theories have gained very rapid development in the last

two decades. In effect, they have significantly extended the applicable range of the conventional linear control systems theory, and is still very much in a rapidly evolving stage.

However, classical mathematics and conventional control theory are quite limited in modeling and controlling complex nonlinear dynamical systems, particularly ill-formulated and partially described physical systems. Fuzzy logic control theory, on the contrary, has shown potential in these kinds of applications. As implied earlier, the main contribution of fuzzy control theory to the field of automation and systems engineering is its ability to handle many practical problems that are difficult, if not impossible, to describe by exact mathematical formulas and to control by conventional techniques. On the other hand, the results of fuzzy control theory are consistent with the classical ones when the conditions of the control system become precise. The motivation for using fuzzy logic technology in control systems also stems from the fact that it allows designers to build a controller even when their understanding of the system is still in a vague, incomplete, and developing phase. It is noteworthy that there are many situations like this in industrial control systems design, which continuously motivates the study of the fuzzy control technology.

2.2 General Structure of Fuzzy Control Systems

Just like other mathematical tools, fuzzy logic, fuzzy set theory, fuzzy modeling, fuzzy control methods, *etc.*, have been developed for solving practical problems. In control systems theory, if the fuzzy interpretation of a real-world problem is correct and if fuzzy theory is developed appropriately, then fuzzy controllers can be suitably designed and they work quite well to their advantages. The entire process is then returned to the original real-world setting, to accomplish the desired system automation. This is what is called the "fuzzification–fuzzy operation–defuzzification" routine in fuzzy control systems practice. The key step — fuzzy operation — is executed by a logical rule base consisting of some IF-THEN rules established by using fuzzy logic and human analysis of the physical problem at hand [4].

2.2.1 Fuzzification

As mentioned earlier, the fundamental feature of fuzzy set theory that distinguishes itself from classical set theory is that it allows partial membership of an element with respect to a set: an element can partially belong to a set and, in the meantime, partially not belong to the same set. For example, Fig. 1 shows different grades of an element, x, belonging to the set, X, specified by several sample (normalized) membership functions, $\mu_X : X \to [0, 1]$. There are two extreme cases: $\mu_X(x) = 0$ means $x \notin X$ and $\mu_X(x) = 1$ means $x \in X$ in the classical sense. But $\mu_X(x) = 0.2$ means x belongs to X only with grade 0.2, or equivalently, x does not belong to X with grade 0.8. A membership function differs from a probability density function in at least two aspects: (i) the area under a membership function curve need not be one (as a matter of fact, it can be infinite); (ii)

when a membership value is, say, 0.2, it does not mean the "chance" to be a member of the set is 20% (since this eventually results in an answer either yes *or* no in probability theory), but rather, it means that 20% of the element belongs to the set (which results in an answer both yes *and* no in fuzzy set theory). Moreover, an element can have more than one membership value at the same time, such as $\mu_X(x) = 0.2$ and $\mu_Y(x) = 0.6$, and they need not be summed to one. The entire setting depends on how large the set X is (or the sets X and Y are) for the associate members, and what kind of shape a membership function should have in order to make sense of the real problem at hand. A set, X, along with a membership function defined on it, $\mu_X(\cdot)$, is called a *fuzzy set* and is denoted (X, μ_X). More examples of fuzzy sets can be seen below, as the discussion continues. This process of transforming a crisp value of an element (say $x = 0.3$) to a fuzzy set (say $x = 0.3 \in X = [0, 1]$ with $\mu_X(x) = 0.2$) is called *fuzzification*.

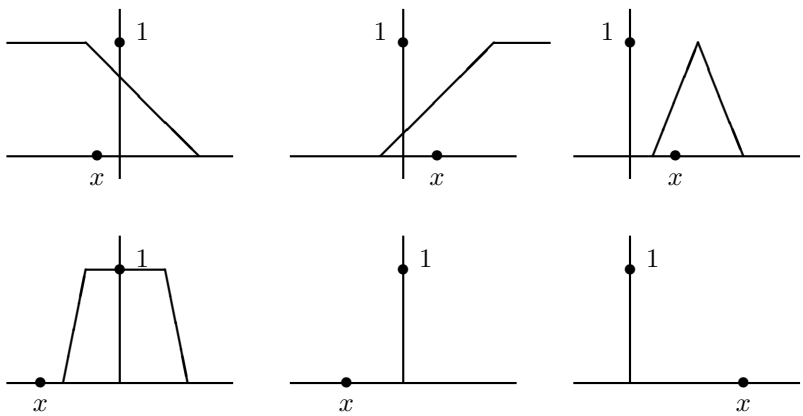


Figure 1: Some examples of membership functions.

Given a classical set of real numbers, $X = [-1, 1]$, a point $x \in X$ assumes a real value, say $x = 0.3$. This is a crisp number without fuzziness. However, if a membership function $\mu_X(\cdot)$ is introduced to associate with the set X, then (X, μ_X) becomes a fuzzy set, and the (same) point $x = 0.3$ has a membership grade quantified by $\mu_X(x)$ (for instance, $\mu_X(x) = 0.9$). As a result, x has not one but two values associated with the point: $x = 0.3$ and $\mu_X(x) = 0.9$. In this sense, x is said to be *fuzzified*. For convenience, instead of saying that "x is in the set X with a membership value $\mu_X(x)$," in common practice it is usually said "x is X," while one should keep in mind that there is always a well-defined membership function associate with the set X. If a member, x, belongs to two fuzzy sets, one says "x is X_1 AND x is X_2," and so on. Here, the relation AND needs a logical operation to perform. As a result, this statement eventually yields only one membership value for the element x, denoted by $\mu_{X_1 \times X_2}(x)$. There are several logical operations to implement the logical AND; they are quite different but all valid within their individual logical system. A commonly used one is to take the minimum value

between the two membership values of the same element x, namely,

$$\mu_{X_1 \times X_2}(x) = \min \{ \mu_{X_1}(x) , \mu_{X_2}(x) \} .$$

2.2.2 Fuzzy logic rule base

The majority of fuzzy logic control systems are knowledge based systems. This means that either their fuzzy models or their fuzzy logic controllers are described by fuzzy logic IF-THEN rules. These rules have to be established based on human expert's knowledge about the system, the controller, and the performance specifications, *etc.*, and they must be implemented by performing rigorous logical operations.

A fuzzy logic rule base is established according to the expert's knowledge about the basic properties and/or fundamental physical as well as mathematical principles of the problem at hand. In much the same way, this is similar to the human decision of what kind of polynomial to use to curve-fit a given set of data when the classical least-squares technique is used for a design.

As mentioned above, a fuzzy rule base incorporates domain expert's knowledge and experience about the problem at hand, and, therefore, is meaningful for the application of interest if it is established appropriately. For example, a car driver knows that if the car moves straight ahead then he does not need to do anything; if the car turns to the right then he needs to steer the car to the left; if the car turns to the right by too much then he needs to take a stronger action to steer the car to the left much more, and so on. This is some basic expert knowledge about car-driving. Here, "much" and "more" *etc.* are fuzzy concepts that have to be quantified by membership functions (see Fig. 2, where part (a) is an example of the description "to the left"). The collection of all such "if $\cdots$ then $\cdots$" principles constitutes a *fuzzy logic rule base* for the problem under investigation. A more precise rule base for set-point tracking control applications is given in the next section, which serves as an introduction to a general model-free approach for fuzzy logic controller design.

To this end, it is helpful to briefly summarize the experience of the driver in the following simplified rule base: Let $X = [-180^o, 180^o]$, x be the position of the car, $\mu_{\text{left}}(\cdot)$ be the membership function for the moving car turning "to the left," $\mu_{\text{right}}(\cdot)$ the membership function for the car turning "to the right," and $\mu_0(\cdot)$ the membership function for the car "moving straight ahead." Here, simplified statements are used, for instance, "x is X_{left}" means "x belongs to X with a membership value $\mu_{\text{left}}(x)$" *etc.* Also, similar notation for the control action, u, of the driver is employed. Then, a simple, yet typical, rule base for this car-driving task is written as

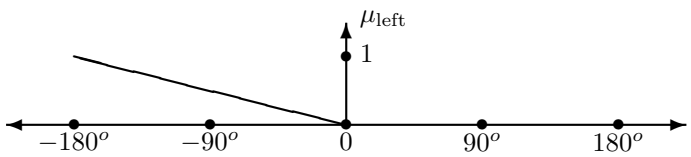
$$\begin{array}{lllllllll} R^{(1)}: & \text{IF} & x & \text{is} & X_{\text{left}} & \text{THEN} & u & \text{is} & U_{\text{right}}; \\ R^{(2)}: & \text{IF} & x & \text{is} & X_{\text{right}} & \text{THEN} & u & \text{is} & U_{\text{left}}; \\ R^{(3)}: & \text{IF} & x & \text{is} & X_0 & \text{THEN} & u & \text{is} & U_0; \end{array}$$

where X_0 means moving straight ahead (not left nor right), as described by the membership function shown in Fig. 2 (c), and "u is U_0" means $u = 0$ (no control action) with a

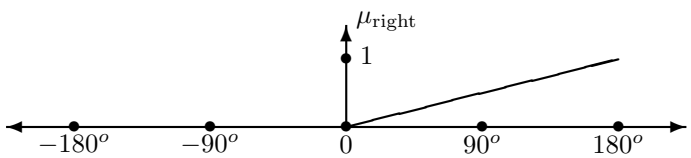
certain grade (if this grade is 1, it means absolutely no control action). Of course, this description only illustrates the basic idea, which is by no means a complete and effective design for a real car-driving applications. In general, a rule base of r rules has the form

$$R^{(k)}: \quad \text{IF } x_1 \text{ is } X_{k1} \text{ AND } \cdots \text{ AND } x_m \text{ is } X_{km} \text{ THEN } u \text{ is } U_k, \tag{1}$$

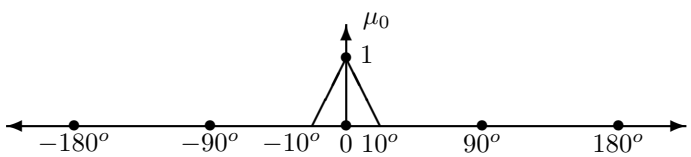
where $m \geq 1$ and $k = 1, \cdots, r$.



(a) Membership function for "to the left."



(b) Membership function for "to the right."



(c) Membership function for "straight ahead."

Figure 2: Membership functions for directions of a moving car.

2.2.3 Defuzzification

As mentioned above, an element of a fuzzy set may have more than one membership values. In Fig. 2, for instance, if $x = 5^o$ then it has two membership values: $\mu_{\text{right}}(x) = 5/180 \approx 0.28$ and $\mu_0(x) = 0.5$. This means that the car is moving to the right by a little, which can also be considered to be moving more or less straight ahead. According to the above-specified rule base, the driver will take two control actions simultaneously, which is unnecessary (physically, impossible). Thus, what should the control action be? To simplify this discussion, suppose that the control action is simply $u = -x$ with the same membership functions $\mu_X = \mu_U$ for all cases. Then, in this case, a natural and realistic control action for the driver to take is a compromise between the two required actions. There are several possible compromise (or, average) formulas that can be used for this purpose, among which the most commonly adopted one that works well in most cases is

the following weighted average formula

$$u = \frac{\mu_{\text{right}}(u) \cdot u + \mu_0(u) \cdot u}{\mu_{\text{right}}(u) + \mu_0(u)} = \frac{0.28 \times (-5^o) + 0.5 \times (-5^o)}{0.28 + 0.5} = -\,0.5^o\,.$$

Here, the result is interpreted as "the driver should turn the car to the left by 5°." This results in a single crisp value for the control action, and this averaging of outputs is called *defuzzification*. The result of defuzzification usually is a physical quantity acceptable by the original real system. Whether or not this defuzzification result works well depends on the correctness and effectiveness of the rule base, while the latter depends on the designer's knowledge and experience about the physical system or process for control. Just like any of the classical design problems, there is generally no unique solution for a problem; an experienced designer usually comes out with a better design.

A general weighted average formula for defuzzification is the following convex combination of the individual outputs:

$$\text{output} = \sum_{i=1}^{r} \alpha_i\, u_i := \sum_{i=1}^{r} \frac{w_i}{\sum_{i=1}^{r} w_i}\; u_i\,, \tag{2}$$

with notation referred to the rule base (1), where

$$w_i = \mu_{U_i}(u_i)\,, \qquad \alpha_i := \frac{w_i}{\sum_{i=1}^{r} w_i} \geq 0\,, \qquad i = 1, \cdots, r\,, \qquad \sum_{i=1}^{r} \alpha_i = 1\,.$$

Sometimes, depending on the design or application, the weights are

$$w_i = \prod_{j=1}^{m} \mu_{X_{ij}}(x_j)\,, \qquad i = 1, \cdots, r\,,$$

which may actually yield similar averaged results in general.

The overall structure of a fuzzy logic controller is shown in Fig. 3.

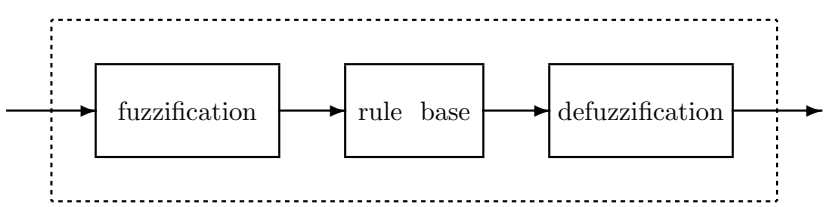


Figure 3: A typical fuzzy logic controller.

3 Some Basic Fuzzy Control Approaches

In this section, fuzzy logic controller design from a model-free approach (without using a mathematical model of the system to be controlled) and from a model-based approach are discussed. Usually, these fuzzy controllers can be used to directly replace a conventional control scheme in a control loop, so as to perform the control actions independently.

3.1 A Model-Free Approach

This general approach of fuzzy logic control works for trajectory tracking for a conventional, even complex, dynamical system that does not have a precise mathematical model.

The basic setup is shown in Fig. 4, where the plant is a conventional system without a mathematical description and all the signals (the setpoint sp, output $y(t)$, control $u(t)$, and error $e(t) = \text{sp} - y(t)$) are crisp. The objective here is to design a controller to achieve the goal $e(t) \to 0$ as $t \to \infty$, without any mathematical formula of the plant except for the assumption that its inputs and outputs are measurable by sensors on line.

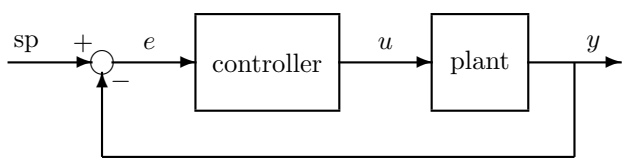


Figure 4: A typical setpoint tracking control system.

If the mathematical formulation of the plant is unknown, how can one develop a controller to control this plant? Fuzzy logic approach turns out to be advantageous in this situation, since it does not need mathematical description about the plant to complete the design of a working controller: it only uses the plant inputs and outputs (but not the state variables, nor any other information) which are usually available through sensors on line. After the design is completed, the entire dashed-block in Fig. 3 is used to replace the "controller" block in Fig. 4.

The fuzzification module transforms the physical values of the current process signal (namely, the error signal e in Fig. 4, see also Fig. 3) into a fuzzy set consisting of an interval of real numbers (for the value-range of the input signals) and a membership function which describes the grades of belongings of the input signals to this interval, at each instant of the control process. The purpose of this fuzzification unit is to make the input physical signal compatible with the fuzzy logic control rules located in the core of the controller. Here, the interval and membership function are both chosen by the designer according to his knowledge about the nature and properties of the given problem, as emphasized previously. This, again, is similar to the decision of a control

engineer as what kind of controller to use in a conventional design (linear or nonlinear, how high the order of a chosen linear controller, what structure of a chosen nonlinear controller, *etc.*). For example, if the physical setpoint is the degree of temperature, say 45^oF, and if the designer knows that the distribution of the error temperature signal, $e(t) = 45^o - y(t)$, is within the range $X = [-20^o, 55^o]$, and the scale of control is required to be in the unit of 1°, then the membership functions for the error signal to be "negative large" (NL), "positive large" (PL), and "zero" (ZO) may be chosen as shown in Fig. 5. Of course, this is by no means a unique choice in a design. Using these membership functions, the controller is expected to eventually drive the output temperature to be within the allowable range: $45^o \pm 1^o$. With these membership functions, when the error signal $e(t) = 5^o$, for instance, it is considered to be "positive large" with membership value one, meaning that the setpoint (45°) is higher than $y(t)$ by too much (since their difference is 5°, as compared to the precision of $\pm 1^o$).

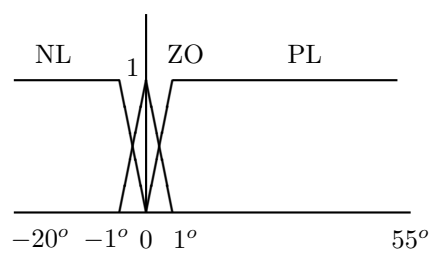


Figure 5: Membership function for the error temperature signal.

The output from the fuzzification module is a fuzzy set consisting of the interval X and three membership functions, μ_{NL}, μ_{PL}, and μ_{ZO}, in this example. The output from fuzzification will be the input to the next module — the fuzzy logic rule base — which only takes fuzzy set inputs to be compatible with the logical IF-THEN rules.

In this temperature setpoint tracking problem, in order to describe the design principle of the rule base, Fig. 6 is helpful. In this figure, if $e > 0$ at a moment, then the setpoint is higher than the output temperature y (since $e = 45^o - y$), which corresponds to two possible situations marked by a and d, respectively. To further distinguish these two situations, another auxiliary input variable is needed. A natural choice is to use the rate of change of the error, $\dot{e}(t) = -\dot{y}(t)$. Here, since the setpoint is a constant, its derivative is zero. Using information from both e and $\dot{e}$, one can completely characterize the changing situation of the output temperature at all times. If, for example, $e > 0$ and $\dot{e} > 0$, then the temperature is currently at situation d rather than situation a, since $\dot{e} > 0$ means $\dot{y} < 0$ which, in turn, signifies that the curve is moving downward.

Based on the above observation from the physical situations of the current temperature against the setpoint, a simple but complete rule base can be established as follows:

$$R^{(1)}: \quad \text{IF} \;\; e(t) > 0 \;\; \text{AND} \;\; \dot{e}(t) > 0 \;\; \text{THEN} \;\; u(t+1) = -u(t)$$
$$R^{(2)}: \quad \text{IF} \;\; e(t) > 0 \;\; \text{AND} \;\; \dot{e}(t) < 0 \;\; \text{THEN} \;\; u(t+1) = \; u(t)$$

Figure 6: Temperature set-point tracking example.

$$
\begin{array}{lll}
R^{(3)}: & \text{IF} \;\; e(t) < 0 \;\; \text{AND} \;\; \dot{e}(t) > 0 \;\; \text{THEN} \;\; u(t+1) = \; u(t) \\
R^{(4)}: & \text{IF} \;\; e(t) < 0 \;\; \text{AND} \;\; \dot{e}(t) < 0 \;\; \text{THEN} \;\; u(t+1) = -u(t)\,.
\end{array}
$$

Here, for illustration purpose, the simplified notation $u(t+1)$ is used to denote the next-step operation in the control, with a normalized sampling unit; it will take a specific form in a real design.

In the above, the first two rules are understood as follows:

$R^{(1)}$: $e > 0$ and $\dot{e} > 0$. As analyzed above, the temperature curve is currently at situation d, so the controller has to change its moving direction to the opposite by changing the current control action to the opposite (since the current control action is driving the output curve downward).

$R^{(2)}$: $e > 0$ but $\dot{e} < 0$. The current temperature curve is at situation a, so the controller does not need to do anything (since the current control action is driving the output curve up toward the set-point).

The other rules can be understood in the same way. The switching control actions may be described, via a change of control action, Δu, as follows:

$$u(t+1) = u(t) + \Delta u(t)$$

together with

$$\Delta u(t) = \begin{cases} 0 & \text{if} \;\; \text{rule } R^{(2)} \text{ is executed} \\ -2u(t) & \text{if} \;\; \text{rule } R^{(1)} \text{ is executed.} \end{cases}$$

Furthermore, to distinguish "positive large" from just "positive" for $e > 0$, one may use those membership functions shown in Fig. 5. Since the error signal, $e(t)$, is fuzzified in the fuzzification module, one can similarly fuzzify the auxiliary signal, $\dot{e}(t)$, in the fuzzification module. Thus, there are two fuzzified inputs, e and $\dot{e}$, for the controller,

and they both have corresponding membership functions describing their properties as "positive large" (μ_{PL}), "negative large" (μ_{NL}), or "zero" (μ_{ZO}). These are shown in Fig. 5. Thus, for the first rule, $R^{(1)}$, one may replace it by a set of more detailed rules as follows:

$$\begin{aligned}
R_1^{(1)}: &\quad \text{IF } e(t) = PL \text{ AND } \dot{e}(t) = PS \text{ THEN } \Delta u(t) = \Delta u_1(t) \\
R_2^{(1)}: &\quad \text{IF } e(t) = PL \text{ AND } \dot{e}(t) = PL \text{ THEN } \Delta u(t) = \Delta u_2(t) \\
R_3^{(1)}: &\quad \text{IF } e(t) = PS \text{ AND } \dot{e}(t) = PS \text{ THEN } \Delta u(t) = \Delta u_3(t) \\
R_4^{(1)}: &\quad \text{IF } e(t) = PS \text{ AND } \dot{e}(t) = PL \text{ THEN } \Delta u(t) = \Delta u_4(t)\,,
\end{aligned}$$

where $e(t) = PL$ means e is "positive large," which is quantified by the membership value $\mu_{PL}(e(t))$, and $\Delta u_1(t)$ may be chosen to be $-2u(t)$ as before, while $\Delta u_2(t)$ may be chosen to be $-u(t)$, and so on. In this way, the initial rule base is enhanced and extended.

In the defuzzification module, new membership functions are needed for the change of the control action, $\Delta u(t)$, if the enhanced rule base described above is used. This is because both the error and the rate of change of the error signals have been fuzzified to be "positive large" or "positive small," the control actions have to be fuzzified accordingly (to be "large" or "small"). Figure 7 shows a typical membership function for Δu, which will be used for defuzzification.

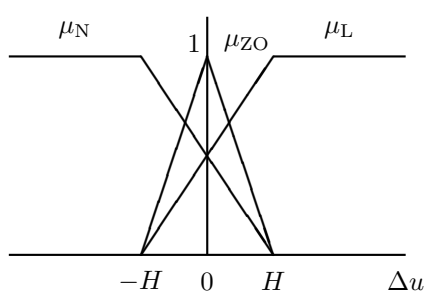


Figure 7: Membership function for the control action.

Now, suppose that a complete, enhanced fuzzy logic rule base has been established:

$$R^{(k)}: \quad \text{IF } e(t) \text{ is } E_k \text{ AND } \dot{e}(t) \text{ is } F_k \text{ THEN } \Delta u_k(t) \text{ is } U_k\,; \quad k = 1, \cdots, \ell\,,$$

where simplified notation, defined earlier, have been used. Then, in the defuzzification module, the weighted average formula can be used to obtain a single crisp value as the control action output from the controller (see Fig. 3):

$$\Delta u(t) = \frac{\sum_{i=1}^{\ell} \mu_{U_i}(\Delta u_i(t)) \cdot \Delta u_i(t)}{\sum_{i=1}^{\ell} \mu_{U_i}(\Delta u_i(t))}\,.$$

This is an average value of the multiple control signals, and is physically meaningful to the given plant.

3.2 A Model-Based Approach

Fuzzy logic controllers can be designed even without any information about the structure of the system for setpoint tracking problems, provided that the system input-outputs (but not the states) can be measured and used on-line, as previously discussed. Note that input-output information is also essential for many conventional system identification techniques [7], which can be obtained through sensors.

If a mathematical model of the system, or a fairly good approximation of it, is available, one may be able to design a fuzzy logic controller with better results such as performance specifications and guaranteed stability. This constitutes a model-based fuzzy control approach. For instance, a locally linear fuzzy system model is described by a rule base of the following form:

$$R_S^{(k)}: \quad \text{IF } x_1 \text{ is } X_{k1} \text{ AND } \cdots \text{ AND } x_m \text{ is } X_{km} \text{ THEN } \dot{\mathbf{x}} = A_k\mathbf{x} + B_k\mathbf{u}\,, \tag{3}$$

where $\{A_k\}$ and $\{B_k\}$ are given constant matrices, $\mathbf{x} = [x_1 \ \cdots \ x_m]^\top$ is the state vector, and $\mathbf{u} = [u_1 \ \cdots \ u_n]^\top$ is a controller to be designed, with $m \geq n \geq 1$, and $k = 1, \cdots, r$. A typical example is the following locally linear single-input system in a canonical form:

$$R_S^{(k)}: \quad \text{IF } x(t) \text{ is } X_{k1} \text{ AND } \cdots \text{ AND } x^{(m)} \text{ is } X_{km} \text{ THEN } \dot{\mathbf{x}} = A_k\mathbf{x} + \mathbf{b}_k\, u\,,$$

where $x^{(j)}(t) = d^j x(t)/dt$, $j = 1, \cdots, m-1$,

$$A_k = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & o & 0 & \cdots & 1 \\ a_{k1} & a_{k2} & a_{k3} & \cdots & a_{km} \end{bmatrix}, \qquad \mathbf{b}_k = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ b_k \end{bmatrix},$$

and $\{a_{ki}\}_{i=1}^m$, $k = 1, \cdots, r$, are known constants.

The fuzzy system model (3) may be rewritten in a more compact form as follows:

$$\dot{\mathbf{x}} = \sum_{i=1}^{r} \alpha_i \left(A_k\mathbf{x} + \mathbf{B}_k\mathbf{u}\right) = \mathbf{A}(\mu(\mathbf{x}))\,\mathbf{x} + \mathbf{B}(\mu(\mathbf{x}))\,\mathbf{u}\,, \tag{4}$$

where $\mu(\mathbf{x}) = \{\mu_{X_{ij}}\}_{i,j=1}^m$. Due to the involvement of the state variables in the membership functions, which appear in both A and B, the formally linear system (4) is generally nonlinear and time-varying in nature. But for some special choices of membership functions, both A and B may actually turn out to be constant, so that the system is indeed linear and time-invariant.

Based on this fuzzy model, (3) or (4), a fuzzy controller, $\mathbf{u}(t)$, can be designed by using some conventional techniques. For example, if a negative state-feedback controller is preferred, then one may design a controller described by the following ruse base:

$$R_C^{(k)}: \quad \text{IF } x_1 \text{ is } X_{k1} \text{ AND } \cdots \text{ AND } x_m \text{ is } X_{km} \text{ THEN } \mathbf{u}(t) = -\,K_k\mathbf{x}(t)\,, \tag{5}$$

where $\{K_k\}_{k=1}^r$ are constant control gain matrices to be determined, $k = 1, \cdots, r$. Thus, the closed-loop controlled system (4) together with (5) becomes

$$R_{SC}^{(k)}: \quad \text{IF } x_1 \text{ is } X_{k1} \text{ AND } \cdots \text{ AND } x_m \text{ is } X_{km} \text{ THEN } \dot{\mathbf{x}} = [A_k - K_k]\,\mathbf{x}\,. \tag{6}$$

For this feedback controlled system, the following is a typical stability condition:

Theorem 1. *If there exists a common positive definite and symmetric constant matrx P such that*

$$A_k^\top P + P A_k = -\,Q\,, \qquad \text{for some } \; Q > 0 \quad \text{for} \quad \text{for all } k = 1, \cdots, r\,,$$

then the fuzzy controlled system (6) is asymptotically stable about zero.

This theorem provides a basic (sufficient) condition for the asymptotic stability of the zero equilibrium solution of the global fuzzy control system as a whole, which can also be viewed as a criterion for tracking control of the system trajectory to the zero set-point. Clearly, stable control gain matrices $\{K_k\}_{k=1}^r$ may be determined according to this criterion in a design.

Sometimes, the above fuzzy model (3) is not available in applications, namely, there is no complete knowledge about the local linear system matrices $\{A_k, B_k\}_{k=1}^r$, except some time-series data obtained from the underlying system. In this case, fuzzy system identification, or *fuzzy system modeling*, becomes necessary. This important topic is further discussed in Section 4 below. Before that, a brief introduction of a (direct) adaptive control approach is in order.

3.3 Adaptive Fuzzy Control

Only model-based adaptive control is briefly discussed in this subsection. In a direct adaptive fuzzy controller, parameters are directly adjusted according to some adaptive law, to reduce (ideally eliminate) the difference between the output of the plant and that of the reference model. Parameters in such a fuzzy controller are those of the membership functions and/or of the rules given in the fuzzy system. In adaptive control, these parameters are automatically tuned during the control process by an adaptation law.

A direct adaptive fuzzy controller can be designed in three steps: (i) determine some fuzzy sets whose membership functions cover the entire operational space for the required control; (ii) use some fuzzy IF-THEN rules to construct an initial rule base for the controller, in which some parameters are free to change; (iii) develop an adaptive law, based on the Lyapunov stability theory for control and stabilization, to adjust the free parameters [17, 18].

Rewrite the fuzzy controller (5) as

$$u(\mathbf{x}) = \sum_{i=1}^{r} \alpha_i\, g_i(\mathbf{x}) = \underline{\alpha}^\top \mathbf{g}(\mathbf{x})\,, \tag{7}$$

with $\underline{\alpha} = [\alpha_1 \ \cdots \ \alpha_r]^\top$ and basis functions $\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}) \ \cdots \ g_r(\mathbf{x})]^\top$, where

$$g_i(\mathbf{x}) = \frac{\prod_{j=1}^{m} \mu_{X_{ij}}(\mathbf{x})}{\sum_{i=1}^{r} \prod_{j=1}^{m} \mu_{X_{ij}}(\mathbf{x})}\,, \qquad i = 1, \cdots, r\,, \tag{8}$$

in which $\{\mu_{X_{ij}}\}$ are the chosen membership functions. Such a fuzzy system can be used as a basic structure for the design of an adaptive controller.

To illustrate the construction of an adaptive fuzzy logic controller, consider a general single-input single-output system,

$$x^{(m)} = f(\mathbf{x}) + b\,u\,, \tag{9}$$

where f is an unknown (linear or nonlinear) function, $b > 0$ is an unknown constant, $\mathbf{x} = [x \ \dot{x} \ \cdots \ x^{(m-1)}]^\top$ is the state vector (assumed to be measurable), and $u(t)$ is the controller to be designed. The objective is to determine a fuzzy controller, $u = u(\mathbf{x}; \underline{\alpha})$, and the associate adaptive rules for adjusting $\underline{\alpha}$, such that

(i) the closed-loop system is globally bounded-input bounded-output stable:

$$\sup_{0 \le t < \infty} ||\mathbf{x}||_2 \le c_x < \infty\,,$$
$$||\underline{\alpha}||_2 \le c_a < \infty\,,$$
$$\sup_{0 \le t < \infty} |u(\mathbf{x}; \underline{\alpha})| \le c_u < \infty\,,$$

for some acceptable bounds, c_x, c_a, and c_u;

(ii) the steady-state tracking error

$$e_{\rm ss} := \lim_{t \to \infty} |e(t)| = \lim_{t \to \infty} |\tilde{x}(t) - x(t)|$$

is minimized, or less than a given tolerance, $\varepsilon > 0$, for a given target, $\tilde{x}(t)$ (which may be a constant set-point).

For this purpose, consider a basic controller, $u_{\rm c} = u_{\rm c}(\mathbf{x}; \underline{\alpha})$, taken to be a fuzzy controller in the form of (7), and a supervisory controller, $u_{\rm s}$, to be determined later. Let the overall controller be

$$u = u_{\rm c} + u_{\rm s} = u_{\rm c}(\mathbf{x}; \underline{\alpha}) + u_{\rm s}(\mathbf{x})\,. \tag{10}$$

Combining (9) and (10) yields

$$x^{(m)} = f(\mathbf{x}) + b\,(\,u_{\rm c}(\mathbf{x}; \underline{\alpha}) + u_{\rm s}(\mathbf{x})\,)\,. \tag{11}$$

After some manipulations, the tracking error dynamics of the closed-loop control system is obtained as

$$\dot{\mathbf{e}} = A_{\mathbf{e}}\,\mathbf{e} + \mathbf{b}_{\rm c}\,(\,u^* - u_{\rm c}(\mathbf{x}; \mathbf{a}) - u_{\rm s}(\mathbf{x})\,)\,, \tag{12}$$

where

$$A_{\mathbf{e}} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_m & -a_{m-1} & \cdots & \cdots & -a_2 & -a_1 \end{bmatrix}, \quad \mathbf{b}_{\mathrm{c}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_1 \\ \vdots \\ a_{m-1} \\ a_m \end{bmatrix},$$

and

$$u^* = \frac{1}{b}\Big(- f(\mathbf{x}) + \tilde{x}^{(m)} + \mathbf{a}^\top \mathbf{e} \Big). \tag{13}$$

Under the condition that $A_{\mathbf{e}}$ has all its eigenvalues with negative real parts, the Lyapunov equation

$$A_{\mathbf{e}}^\top P + P\, A_{\mathbf{e}} = -\, Q < 0$$

has a unique positive definite and symmetric matrix solution, P, for a given positive definite and symmetric matrix Q (usually, the identity matrix $Q = I_m$ is convenient to use). Using this matrix P, the design of the entire controller is accomplished as follows:

(1) *Design of the supervisory controller* u_{s}

Define a Lyapunov function candidate, $V_{\mathbf{e}} = \frac{1}{2}\,\mathbf{e}^\top P\,\mathbf{e}$, where $P > 0$ is the matrix solution obtained from the above Lyapunov equation. Then, to ensure

$$\begin{aligned} \dot{V}_{\mathbf{e}} &= -\frac{1}{2}\,\mathbf{e}^\top Q\,\mathbf{e} + \mathbf{e}^\top P\,\mathbf{b}_{\mathrm{c}}\,(\,u^* - u_{\mathrm{c}}(\mathbf{x};\underline{\alpha}) - u_{\mathrm{s}}(\mathbf{x})\,) \\ &\le -\frac{1}{2}\,\mathbf{e}^\top Q\,\mathbf{e} + |\,\mathbf{e}^\top P\,\mathbf{b}_{\mathrm{c}}\,| \cdot (\,|u^*| + |u_{\mathrm{c}}(\mathbf{x};\underline{\alpha})|\,) - \mathbf{e}^\top P\,\mathbf{b}_{\mathrm{c}}\,u_{\mathrm{s}}(\mathbf{x}) \\ &< 0\,, \end{aligned}$$

a supervisory controller can be chosen as

$$u_{\mathrm{s}}(\mathbf{x}) = S_{\mathrm{v}}\,\mathrm{sgn}\,[\mathbf{e}^\top P\,\mathbf{b}_{\mathrm{c}}]\Big(\,|u_{\mathrm{c}}| + \beta^{-1}(f_b(\mathbf{x}) + |\tilde{x}^{(m)}| + |\mathbf{k}^\top \mathbf{e}|\,)\Big), \tag{14}$$

where β is a constant satisfying $0 < \beta \le b$, f_b is a predetermined function satisfying $|f(\mathbf{x})| \le f_b(\mathbf{x})$ for all $\mathbf{x}$, with

$$\mathrm{sgn}\,[z] = \begin{cases} 1 & \text{if} \quad z > 0 \\ 0 & \text{if} \quad z = 0 \\ -1 & \text{if} \quad z > 0 \end{cases} \qquad \text{and} \qquad S_{\mathrm{v}} = \begin{cases} 1 & \text{if} \quad V_{\mathbf{e}} > \gamma_{\mathrm{v}} \\ 0 & \text{if} \quad V_{\mathbf{e}} \le \gamma_{\mathrm{v}}\,, \end{cases}$$

for some constant $\gamma_{\mathrm{v}} > 0$ specified by the control designer. Equation (14) is a supervisory type of controller due to the free choice of S_{v}.

(2) *Design of the basic controller* u_{c}

Let the basic controller be (7), namely,

$$u_{\mathrm{c}}(\mathbf{x};\underline{\alpha}) = \sum_{k=1}^{r} \alpha_k\, g_k(\mathbf{x}) = \underline{\alpha}^\top \mathbf{g}(\mathbf{x})\,. \tag{15}$$

It is necessary to develop an adaptive law to adjust $\underline{\alpha}$. Let the optimal vector be

$$\underline{\alpha}^* = \arg \min_{||\underline{\alpha}||_2 \le c_a} \left(\sup_{||\mathbf{x}||_2 \le c_x} |u_{\mathrm{c}}(\mathbf{x}; \underline{\alpha}) - u^*| \right)$$

and the minimal approximation error be

$$e^* = u_{\mathrm{c}}(\mathbf{x}; \underline{\alpha}^*) - u^* .$$

Then, the error equation is

$$\begin{aligned} \dot{\mathbf{e}} &= A_{\mathbf{e}}\mathbf{e} + \mathbf{b}_{\mathrm{c}} \left(u_{\mathrm{c}}(\mathbf{x}; \underline{\alpha}^*) - u_{\mathrm{c}}(\mathbf{x}; \underline{\alpha}) \right) - \mathbf{b}_{\mathrm{c}}\, u_{\mathrm{s}}(\mathbf{x}) - \mathbf{b}_{\mathrm{c}}\, e^* \\ &= A_{\mathbf{e}}\mathbf{e} + \mathbf{b}_{\mathrm{c}}\, \mathbf{h}^\top \mathbf{g}(\mathbf{x}) - \mathbf{b}_{\mathrm{c}}\, u_{\mathrm{s}} - \mathbf{b}_{\mathrm{c}}\, e^* , \end{aligned} \tag{16}$$

where $\mathbf{h} := \underline{\alpha}^* - \underline{\alpha}$ and the fuzzy basis function is given by (8).

Consider a Lyapunov function candidate,

$$V_{\mathbf{e},\mathbf{h}} = \frac{1}{2}\, \mathbf{e}^\top P\, \mathbf{e} + \frac{b}{2\alpha}\, \mathbf{h}^\top \mathbf{h} ,$$

where $\gamma > 0$ is an adjustable parameter to be determined later. From the above Lyapunov equation and the error equation (16), the derivative of this Lyapunov function is obtained as

$$\begin{aligned} \dot{V}_{\mathbf{e},\mathbf{h}} &= -\frac{1}{2}\, \mathbf{e}^\top Q\, \mathbf{e} + \mathbf{e}^\top P\, \mathbf{b}_{\mathrm{c}} \left(\mathbf{h}^\top \mathbf{g}(\mathbf{x}) - u_{\mathrm{s}} - e^* \right) + \frac{b}{\alpha}\, \mathbf{h}^\top \dot{\mathbf{h}} \\ &= -\frac{1}{2}\, \mathbf{e}^\top Q\, \mathbf{e} + \frac{b}{\gamma}\, \mathbf{h}^\top \left(\gamma\, \mathbf{e}^\top \mathbf{p}_m\, \mathbf{g}(\mathbf{x}) + \dot{\mathbf{h}} \right) - \mathbf{e}^\top P\, \mathbf{b}_{\mathrm{c}} \left(u_{\mathrm{s}} + e^* \right) , \end{aligned}$$

where $\mathbf{p}_m$ is the last column of P, satisfying $\mathbf{e}^\top P\, \mathbf{b}_{\mathrm{c}} = \mathbf{e}^\top \mathbf{p}_m\, b$. If the basic adaptive law

$$\dot{\underline{\alpha}} = \gamma\, \mathbf{e}^\top \mathbf{p}_m\, \mathbf{g}(\mathbf{x}) \tag{17}$$

is used, then since e^* is ideally zero,

$$\dot{V}_{\mathbf{e},\mathbf{h}} \le -\frac{1}{2}\, \mathbf{e}^\top Q\, \mathbf{e} - \mathbf{e}^\top P\, \mathbf{b}_{\mathrm{c}}\, e^* \; < \; 0 .$$

To further guarantee the objective $||\underline{\alpha}|| \le c_a$, a standard projection estimation algorithm can be used to modify the above adaptive law.

Figure 8 shows the overall schematic diagram of the direct adaptive fuzzy control system.

Figure 8: Configuration of a direct adaptive fuzzy control system.

4 Fuzzy System Modeling

Recall that the basic objective of system modeling is to establish an input-output representative mapping that can satisfactorily describe the system behaviors over the entire operational space.

Conventional system modeling techniques suggest to construct a model by using the available input-output data based upon empirical or physical knowledge about the structure and/or order of (non)linearity of the unknown system; which usually leads to the determination of a set of differential or difference equations [7]. This kind of approaches are effective only when the underlying system is relatively simple and mathematically well-defined. They often fail to handle complex, uncertain, vague, ill-defined physical systems because they always try to find a precise function or a fixed structure to fit to the assumed system; unfortunately most real-world problems do not obey such simple, idealized, and subjective mathematical rules. This weakness of the conventional mathematical modeling approaches has been realized for quite a long time, by many experts in the classical control and system engineering communities.

A perfect modeling methodology can never be found; yet a better approach is quite possible. According to the incompatibility principle [20], as the complexity of a system increases, human's ability to make precise and significant statements about its behaviors diminishes, until a threshold is reached beyond which precision and significance become almost mutually exclusive characteristics. Under this principle, Zadeh proposed a modeling method of human thinking with linguistic fuzzy set rather than crisp numbers [19, 20], which eventually leads to the development of various fuzzy modeling techniques later on.

In this section, the important issue of fuzzy system modeling is addressed. Generally

speaking, system modeling involves at least two basic parts: structure identification and parameters identification, which are first discussed below.

4.1 Structure Identification

In structure identification of a fuzzy model, the first step is to select some appropriate input variables from the collection of possible system inputs. The second step is to determine the number of membership functions for each input variable. This process is closely related to the partitioning of input space. There are several types of fuzzy modeling techniques which employ the same antecedent structure. Input space partitioning methods are useful for determination of such structures. Only some most commonly used partitioning methods for fuzzy models are discussed here.

It is well known that a multi-input multi-output system, described by a fuzzy rule base, can be decomposed into a number of multi-input single-output rule bases. Therefore, partitioning methods for the latetr are essential, some of which are briefly introduced below.

(a) **Grid Partitioning**:

Figure 9 (a) shows a typical grid partition in a two-dimensional input space. Fuzzy grids can be used to generate fuzzy rules based on system input-output training data. Also, a one-pass build-up procedure is possible that can avoid the time-consuming learning process [18]. The performance depends heavily on the definition of the grid. In general, the finer the grid is, the better the performance will be. However, it is likely that the fuzzy grid regions used in this approach will not cover all training data, and so some regions remain undefined. Adaptive fuzzy grid partitioning can be used to refine and even optimize this process. In the adaptive approach, a uniformly partitioned grid is first used for initialization. As the process goes on, the parameters in the antecedent membership functions will be adjusted. Consequently, the fuzzy grid evolves. The gradient descent method can then be used to optimize the size and location of the fuzzy grid regions and the overlapping degree among them. For this grid partition method, there is a major drawback: the performance suffers from an exponential explosion of the number of inputs or membership functions as the input variables increase. For example, a fuzzy model has 5 inputs and 5 membership functions associated with each input would result in $5^5 = 3125$ IF-THEN rules. This is referred to as the "curse of dimensionality," and is a common issue for most partitioning methods.

(b) **Tree Partitioning**:

Figure 9 (b) visualizes a tree partition. The tree partitioning results from a series of guillotine cuts. Each region is generated by a guillotine cut, which is made entirely across the subspace to be partitioned. At the $(k-1)$st iteration step, the input space is partitioned into k regions. Then a guillotine cut is applied to one of these regions to further partition the entire space into $k+1$ regions. There are several strategies for deciding which dimension to cut, where to cut at each step, and when to stop.

This flexible tree partitioning algorithm relieves the problem of curse of dimensionality. However, more membership functions are needed for each input variable as a comparison, and these membership functions usually do not have clear linguistic meanings. Moreover, the resulting fuzzy model consequently is less descriptive.

(c) **Scatter Partitioning**:

Figure 9 (c) illustrates a scatter partition. This method extracts fuzzy rules directly from numerical data [1]. Suppose that a one-dimensional output, y, and an m-dimensional input vector, $\mathbf{x}$, are available. First, one divides the output space into n intervals as follows:

$$[y_0, y_1], \quad (y_1, y_2], \quad \cdots (y_{n-1}, y_n],$$

where the ith interval is labeled as "output interval i." Then, activation hyperboxes are determined, which define the input region corresponding to the output interval i, by calculating the minimum and maximum values of the input data for each output interval. If the activation hyperbox for the output interval i overlaps with the activation hyperbox for the output interval j, then the overlapped region is defined as an inhibition hyperbox. If the input data for output intervals i and/or j exist in the inhibition hyperbox, within this inhibition hyperbox one or two additional activation hyperboxes will be defined. Moreover, if two activation hyperboxes are defined and they overlap, an additional inhibition hyperbox is further defined. This procedure is repeated until overlapping is resolved.

(a) (b) (c)

Figure 9: Three typical MISO partitioning methods.

4.2 Parameters Identification

After the system structure has been determined, parameters identification is in order. In this process, the optimal parameters of a fuzzy model that can best describe the input-output behavior of the underlying system are searched. More precisely, parameters

identification means identification of optimal parameters of fuzzy sets in the IF and the THEN parts of each rule by various optimization techniques.

Sometimes, structure and parameters are identified under the same framework through fuzzy modeling. There are virtually many different approaches to modeling a (control) system using the fuzzy set and fuzzy system theories (*e.g.*, [5, 10]); but only the classical least-squares optimization method and the general Genetic Algorithm (GA) optimization technique are introduced in this section. The main reason is that the least-squares method is perhaps the oldest and most popular method for optimization (and, hence, for system modeling) based on measurement data, and the GA optimization approach is very general and effective, competitive with many other successful nonfuzzy types of optimization-based modeling methods such as artificial neural networks and statistical monte carlo, but has some special features that are advantageous for optimal fuzzy system modeling. A good understanding of these two modeling techniques is a pathway to other practical fuzzy system modeling methodologies.

4.2.1 An Approach Using Least-Squares Optimization

Recall the fuzzy controller (7). Note that this is a fuzzy system itself and, indeed, most fuzzy systems possess this typical format. In other words, a general fuzzy system has the following generic form:

$$f(\mathbf{x}) = \sum_{k=1}^{m} \alpha_k g_k(\mathbf{x}) = \underline{\alpha}\, \mathbf{g}_k(\mathbf{x})\,, \tag{18}$$

where $\underline{\alpha} = [\alpha_1 \ \cdots \ \alpha_m]^\top$ are constant coefficients and

$$g_k(\mathbf{x}) = \frac{\prod_{i=1}^{n} \mu_{X_{kj}}(\mathbf{x})}{\sum_{k=1}^{m} \left(\prod_{i=1}^{n} \mu_{X_{kj}}(\mathbf{x}) \right)}\,, \qquad k = 1, \cdots, m\,, \tag{19}$$

are the basis functions, in which $\mu_{X_{kj}}(\cdot)$ are the chosen membership functions.

Usually, this linear combination of fuzzy basis functions is not a perfect model for a physical system or process. So suppose that the system or process output is

$$y(t) = \sum_{k=1}^{m} \alpha_k g_k(\mathbf{x}) + e(t)\,, \tag{20}$$

where $y(t)$ is the system output and $e(t)$ represents the modeling error, which is assumed to be uncorrelated with the fuzzy basis functions $\{g_k(\cdot)\}_{k=1}^{m}$ in this discussion.

For least-squares optimization, n pairs of system input-output data are assumed to be given:

$$(\mathbf{x}_d(t_i), y_d(t_i)), \qquad i = 1, \cdots, n\,.$$

The goal is to find the best possible fuzzy basis functions, such that the total least-squares error between the data set and the system outputs, $\{y(t_i)\}_{i=1}^{n}$, is minimized. To do so,

the linear model (20) is first written in a matrix form over the time domain $t_1 < \cdots < t_n$, namely,

$$\mathbf{y} = G\,\underline{\alpha} + \mathbf{e}\,, \tag{21}$$

where $\mathbf{y} = [y(t_1), \cdots, y(t_n)]^\top$, $\mathbf{e} = [e(t_1), \cdots, e(t_n)]^\top$, and

$$G := [\mathbf{g}_1, \cdots, \mathbf{g}_n]^\top = \begin{bmatrix} g_1(t_1) & \cdots & g_m(t_1) \\ \vdots & & \vdots \\ g_1(t_n) & \cdots & g_m(t_n) \end{bmatrix},$$

with $\mathbf{g}_j = [g_j(t_1), \cdots, g_j(t_n)]^\top$, $j = 1, \cdots, n$.

The first step of least-squares optimization is to transform the set of numbers, $g_i(t_j)$, $i = 1, \cdots, m$ and $j = 1, \cdots, n$, into a set of orthogonal basis vectors, and only significant basis vectors are used to form the final least-squares optimization. Here, the Gaussian membership function

$$\mu_{X_{kj}}(x_k) = c_{kj}\,\exp\left\{-\frac{1}{2}\left(\frac{x_k - \bar{x}_{kj}}{\sigma_{kj}}\right)^2\right\} \tag{22}$$

is used as an example to illustrate the computational algorithm, which can be separated into two steps: the determination of initial fuzzy basis functions and a selection of those significant ones among them.

One approach to initializing the fuzzy basis functions is to choose n initial basis functions, $g_k(\mathbf{x})$, in the form of (19) with $m = n$ in this discussion, and initially with $c_{kj} = 1$, $\bar{x}_{kj} = x_k(t_j)$, and

$$\sigma_{kj} = \frac{1}{m_\ell}\left[\,\max\{x_k(t_j), j = 1, \cdots, n\} - \min\{x_k(t_j), j = 1, \cdots, n\}\,\right],$$

$k = 1, \cdots, n$, where m_ℓ is the number of the basis functions in the final expression, which is determined by the designer based on experience (usually, $m_\ell \ll n$).

After choosing the initial fuzzy basis functions the next step is to select the most significant ones among them. This process is based on the classical Gram-Schmidt orthogonalization, while c_{kj}, $\bar{x}_{kj}$, and σ_{kj} are all fixed. The procedure is as follows:

Step 1. For $j = 1$, compute

$$\begin{aligned} \mathbf{w}_1^{(i)} &= \mathbf{g}_i(\mathbf{x}_d) = [g_i(x_d(t_1)), \cdots, g_i(x_d(t_n))]^\top \\ h_1^{(i)} &= \frac{(\mathbf{w}_1^{(i)})^\top \mathbf{y}_d}{(\mathbf{w}_1^{(i)})^\top \mathbf{w}_1^{(i)}} \\ \varepsilon_{1i} &= (h_1^{(i)})^2 \frac{(\mathbf{w}_1^{(i)})^\top \mathbf{w}_1^{(i)}}{\mathbf{y}_d^\top \mathbf{y}_d} \qquad (1 \le i \le n), \end{aligned}$$

where $\mathbf{x}_d = [x_d(t_1), \cdots, x_d(t_n)]^\top$ and $\mathbf{y}_d = [y_d(t_1), \cdots, y_d(t_n)]^\top$ are the input-output data set. Then, compute

$$\varepsilon_1^{(i_1)} = \max\{\varepsilon_1^{(i)} : 1 \le i \le n\}$$

and let

$$\mathbf{w}_1 = \mathbf{w}_1^{(i_1)} = \mathbf{g}_{i_1} \qquad \text{and} \qquad h_1 = h_1^{(i_1)}.$$

Step 2. For each j $(2 \le j \le m_\ell)$, compute

$$\begin{aligned}
c_{kj}^{(i)} &= \frac{\mathbf{w}_k^\top \mathbf{g}_i}{\mathbf{w}_k^\top \mathbf{w}_k} \qquad (1 \le k \le j) \\
\mathbf{w}_j^{(i)} &= \mathbf{g}_i - \textstyle\sum_{k=1}^{j-1} c_{kj}^{(i)} \mathbf{w}_k \\
h_j^{(i)} &= \frac{(\mathbf{w}_j^{(i)})^\top \mathbf{y}_d}{(\mathbf{w}_j^{(i)})^\top \mathbf{w}_k^{(i)}} \\
\varepsilon_j^{(i)} &= \left(h_j^{(i)}\right)^2 \frac{(\mathbf{w}_j^{(i)})^\top \mathbf{w}_j^{(i)}}{\mathbf{y}_d^\top \mathbf{y}_d} \\
\varepsilon_k^{(i_j)} &= \max\left\{\varepsilon_j^{(i)} : 1 \le i \le n; i \ne i_1, \cdots, i \ne i_{j-1}\right\},
\end{aligned}$$

where $\varepsilon_j^{(i)}$ represents the error-reduction ratio due to $\mathbf{w}_j^{(i)}$. Pick

$$\mathbf{w}_j = \mathbf{w}_j^{(i_j)} \qquad \text{and} \qquad h_k = h_k^{(i_j)}.$$

Step 3. Solve equation

$$A^{(m_\ell)} \underline{\alpha}^{(m_\ell)} = \mathbf{h}^{(m_\ell)}$$

for $\underline{\alpha}^{(m_\ell)} = [\alpha_1^{(m_\ell)}, \cdots, \alpha_{m_\ell}^{(m_\ell)}]^\top$, where $\mathbf{h}^{(m_\ell)} = [h_1, \cdots, h_{m_\ell}]^\top$ and

$$A^{(m_\ell)} = \begin{bmatrix} 1 & c_{12}^{(i_2)} & c_{13}^{(i_3)} & \cdots & c_{1m_\ell}^{(i_{m_\ell})} \\ 0 & 1 & c_{23}^{(i_3)} & \cdots & c_{2m_\ell}^{(i_{m_\ell})} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & c_{m_\ell-1,m_\ell}^{(i_{m_\ell})} \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

The final result for system (18), after the above least-squares fitting to data, is

$$f(\mathbf{x}) = \sum_{k=1}^{m_\ell} \alpha_k^{(m_\ell)} g_{i_k}(\mathbf{x}). \tag{23}$$

4.2.2 An Approach Using Genetic Algorithms

Although it is possible to construct a fuzzy model manually by a designer, the identification procedure of a fuzzy model is generally very tedious for a complex and large-scale system. This stimulates the recent development of automatic fuzzy modeling methods. Among various automatic fuzzy modeling techniques, the genetic algorithm (GA) has some attractive features such as its great flexibility and robust optimization ability [13].

GA was first introduced by Holland [8] and was applied to fuzzy modeling and fuzzy control since the late 1980s. GA can be used to find an optimal or suboptimal fuzzy model to describe a given system without manual design (not even a design of the fuzzy model structure) [11–14]. In addition, GA fuzzy modeling method can be integrated with other components of a fuzzy system, so as to achieve overall superior performance in control and automation.

4.2.2.1 Genetic algorithm preliminaries

GAs are optimization methods in which a stochastic search algorithm is performed based on the basic biological principles of selection, crossover and mutation. A GA algorithm encodes each point in a solution space into a string composing of binary, integer, or real values, called a *chromosome*. Each point is assigned a fitness value from zero to one, which is usually taken to be the same as the objective function to be maximized, although they can be different.

Unlike other optimization methods such as the familiar gradient descent method, a GA scheme keeps a set of points as a *population*, which is evolved repeatedly toward a better and then even better fitness value. In each *generation*, GA generates a new population using genetic operators such as crossover and mutation. Through these operations, individuals with higher fitness values are more likely to survive and to participate in the next genetic operations. After a number of generations, individuals with higher fitness values are kept in the population while the others are eliminated. GAs, therefore, can ensure a gradual increasing of improving solutions, till a desired optimal or suboptimal solution is obtained.

(i) *Basic GA terminologies*

To introduce the GAs, some new terminologies are needed. As mentioned, GAs are mimicking the natural genetics to achieve optimality through random search, and so they employ quite a few terminologies from the natural genetics. Table 1 shows a comparison of natural and GA terminologies.

Table 1: GA Terminologies

Natural genetics	Artificial genetic algorithm
Chromosome	Chromosome, string
Gene	Gene, feature, character, detector
Allele	Allele, feature value
Locus	Locus, string position
Genotype	Genotype, structure
Phenotype	Phenotype, parameter set, alternative solution, decoded structure

(ii) *Basic GA elements*

A simple genetic algorithm (SGA) was first described by Goldberg [6] and is used here for illustration. A pseudo-code outline of a SGA is shown in Table 2, where the population at time t is represented by a time-dependent variable, $P = P(t)$, with an initial population $P(0)$ which can be randomly estimated when the algorithm is run for an application.

Table 2: GA Elements

```
Procedure GA
Begin
t = 0;
Initialize P(t);
Evaluate P(t);
While not finished do
    Begin
    t = t + 1;
    Reproduce P(t) from P(t − 1);
    Crossover individuals in P(t);
    Mutate individuals in P(t);
    Evaluate P(t);
    End
End
```

(iii) **Population representation and initialization**

Individuals, or current approximations, are encoded as *strings* (*i.e.*, chromosomes) composing of some alphabets, so that the *genotypes* (chromosome values) are uniquely mapped onto the decision variable (*phenotype*) domain. The most commonly used representation in GAs is the binary alphabet, $\{0, 1\}$, although other representations can be used (*e.g.*, ternary, integer, real-valued, *etc.* [15]). For example, a problem with two variables, x_1 and x_2, may be mapped onto the chromosome structure in a way as shown in Fig. 10, where x_1 is encoded with 10 bits and x_2 with 15 bits, possibly reflecting the level of accuracy or range of the individual decision variables. Examining the chromosome strings individually yields the information needed for solving the problem at hand.

The search process, described below, will operate on this encoding decision variables rather than the decision variables themselves, except when real-valued genes are used. After a representation method has been chosen to use, the first step in the SGA is to

Figure 10: Structure of binary chromosome.

create an initial population. This is usually achieved by generating the required number of individuals by using a random number generator, which uniformly distributes initial numbers in the desired range. For example, with a binary population of N individuals whose chromosomes are L bits long, $L \times L$ random numbers would be generated and uniformly distributed over the interval $[0, 1]$.

(iv) *Objective and fitness functions*

The *objective function* is used to measure the performance of the individuals over the problem domain. For a minimization problem, the mostly fit individuals will have the lowest numerical value of the associated objective function. This raw measure of fitness is usually used as an intermediate stage in determining the relative performance of individuals in a GA.

Another function, the *fitness function*, is normally used to transform the objective function value into a measure of relative fitness. Mathematically,

$$F(x) = g(f(x)) \,,$$

where f is the objective function, g the transform that maps the value of f to a nonnegative number, and F is the resulting relative fitness. This mapping is always necessary when the objective function is to be minimized, since the lower objective function values usually indicates better-fit individuals. In general, the fitness function value corresponds to the number of offspring, and an individual can expect to produce this value in the next generation. A commonly used transform is that of proportional fitness assignment, defined by the individual's raw performance, $f(x_i)$, relative to the whole population, namely,

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{N} f(x_i)} \,,$$

where N is the population size and x_i is the phenotypic value of individual i, $i = 1, \cdots, N$.

Although the above fitness assignment ensures that each individual has a certain probability of reproduction according to its relative fitness, it does not account for negative objective function values. A linear transform, which offsets the objective function, is often used prior to the fitness assignment. It takes the form

$$F(x) = a\, f(x) + b \,,$$

where a is a positive scaling factor if the optimization is to maximize the objective function but is negative if it is a minimization, and the offset b is used to ensure that the resulting fitness values are all negative.

Then, the selection algorithm selects individuals for reproduction on the basis of their relative fitness. Using linear scaling, the expected number of offspring is approximately proportional to that of good individual's performance.

(v) *Reproduction*

Once each individual has been assigned a fitness value, they can be chosen from the population with a probability according to their relative fitness. They can then be recombined to produce the next generation. Genetic operators directly manipulate the character (genes) of the chromosomes, based on the assumption that certain individuals' gene codes produce better-fit individuals on average. Most widely used genetic operators in GA are selection, crossover, and mutation operators. They are often run simultaneously in an GA program, which are separately discussed below.

(vi) *Selection*

Selection is the process of determining the number of times, or trials, a particular individual is chosen for reproduction. Thus, it is the number of offspring that an individual will produce in the *mating pool*, a temporary population where crossover and mutation operations are applied to each individual. The selection of individuals has two separate processes:

(1) determination of the number of trials an individual can expect to receive;

(2) conversion of the expected number of trials into a discrete number of offspring.

The first part is concerned with the transform of raw fitness values into a real-valued expectation of an individual's reproduction probability, which was used as the fitness assignment before. The second part is the probabilistic selection of individuals for reproduction based on the fitness of individuals relative to one another, and is sometimes termed *sampling*.

The widely used selection methods in the literature are the roulette wheel selection. When this method is applied, individual is assigned its own territory in a roulette wheel proportional to its fitness value. After assigning a territory to each individual, individuals are selected by wheel spinning until a mating pool is constructed. In roulette wheel selection, individual with higher fitness value has higher probability of leaving its offspring in the mating pool.

As an simple example, consider a hypothetical population of five with binary coding. Table 3 shows the population and its corresponding fitness values. In this example, the total fitness of population is 1. Each individual has its own territory in the roulette wheel as shown in Fig. 11. That is, the chromosome 4 has a 29.6634% chance of being copied into the mating pool (*i.e.*, being selected as a parent).

Table 3: Chromosomes and Fitness Values in the Example

Chromosome	Fitness value
01010	0.037008
01101	0.041065
10111	0.296705
10110	0.296634
11011	0.328589

Figure 11: Roulette wheel with assigned individuals.

(vii) *Crossover (recombination)*

The crossover operator defines the procedure for generating children from two parents. Analogous to biological crossover, it exchanges genes at a randomly selected crossover point from also randomly selected parents in the mating pool to generate children. One popular method is the following: Parent chromosomes are cut at randomly selected points, which can be more than one, to exchange their genes at some specified crossover points with a user-specified crossover probability. This crossover method is categorized into single-point crossover and multi-point crossover according to the number of crossover points. Figure 12 shows a single-point crossover and Fig. 13 shows a multi-point crossover. Another form of crossover is uniform crossover, where parents are also randomly selected in the mating pool. Crossover operation is applied with the crossover probability, and each gene in one children are selected from both parents with a user-specified probability. In the other children, a chromosome is constructed by selecting from both parents some genes that are not selected in the first children. Uniform crossover often works well with small populations of chromosomes and for simpler problems [14]. Figure 14 shows the operation of uniform crossover.

Figure 12: Single-point crossover.

Figure 13: Multi-point crossover.

Figure 14: Uniform crossover.

(viii) *Mutation*

In natural biological genetics, mutation is a process where one allele value of a gene is replaced by another to produce a new genetic structure. In GAs, mutation operation is randomly applied to individuals, so as to change their gene value with a mutation probability, P_m, which is very low in general. Figure 15 visualizes the mutation operation, where the fifth gene value in the parent chromosome is changed from 1 to 0.

Figure 15: Mutation operator.

(ix) *GA Parameters*

The choice of the mutation probability P_m and the crossover probability P_c as two control parameters can be a complex nonlinear optimization problem. Their settings are critically dependent upon the nature of the objective function. This selection issue still remains open to better resolutions. One suggestion is that for large population size (say 100), crossover rate is 0.6 and mutation rate is 0.001, while for small population size (such as 30), crossover rate is 0.9 and mutation rate is 0.01 [21].

4.2.2.2 GA-based fuzzy system modeling

In a GA, parameters for the given problem are represented by the chromosome. This chromosome may contain one or more substrings. Each chromosome, therefore, contains a possible solution to the problem. Fitness function is used to evaluate how well a

chromosome solves the problem. In the GA-based approach for fuzzy modeling, each chromosome represents a specific fuzzy model, and the ultimate goal is to carefully design a good (ideally optimal) chromosome to represent a desired fuzzy model.

As discussed earlier, structure and parameters of a system model are two basic issues in constructing a suitable fuzzy model. In the GA-based approach, this information is incorporated into a chromosome with a specific structure.

Chromosome stuctures

As an example, consider a simple fuzzy model with only one rule, along with the scatter partition which is to be encoded to a chromosome. Suppose that both real number coding and integer number coding are used. The parameters and the structure of the fuzzy model are encoded into one or more substrings in the chromosome. A chromosome is composed of two substrings (candidate substring and decision substring) and these substrings are divided into two parts (IF part and THEN part), as shown in Fig. 16

Figure 16: A chromosion structure for fuzzy modeling.

The candidate substring is encoded by real numbers, as shown in Fig. 17. It contains the candidates for the parameters of a membership function in the IF part, and the fuzzy singleton membership function in the THEN part. Figure 17 describes the coding format of the candidate substring in a chromosome, where n is the number of input variables, r the number of candidates for parameters in the IF part, and s the number of candidates for the real numbers in the THEN part.

Decision substrings are encoded by integers. These substrings determine the structure of rules and the number of rules by choosing one of the parameters in the candidate substrings, as illustrated by Fig. 18.

The decision substrings for the IF part determine the premise structure of the fuzzy rule base. It is composed of n genes that take integer values (alleles) between 0 and r. According to this value, an appropriate parameter in the candidate substring is selected. A zero value means that the related input is not included in the rule. A decision substring for the THEN part is composed of c (the maximum number of rules) genes that take the integer values between 0 and s, which choose appropriate values from the candidate

Figure 17: The candidate substrings in a chromosome.

Figure 18: The decision substrings in a chromosome.

substring for the THEN part. In this substring, the gene taking the zero value deletes the related rule. Therefore, these substrings determine the structure of the THEN part and the number of rules. Figure 19 illustrates an example of decoding the chromosome, with the resulting fuzzy rule shown in Fig. 20.

Figure 19: An example of genetic decoding process.

The fitness function

To measure the performance of the GA-based fuzzy modeling, an objective function is defined for optimization. This objective function is chosen by the designer, usually is a least-squares matching measure of the form

$$J = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \tilde{y}_i \right)^2 , \tag{24}$$

where $\{y_i\}$ and $\{\tilde{y}_i\}$ are the fuzzy model outputs and desired outputs, respectively, and n is the number of the data used.

Figure 20: The first fuzzy rule obtained by the decoding process.

Since GA is guided by the fitness values and requires literally no limit on the formulation of its performance measure, one can incorporate more information about a fuzzy model into a fitness function:

$$f = g(J_{\text{structure}}, J_{\text{accuracy}}, \cdots)$$

One example of a fitness function is

$$f(J) = \frac{\lambda}{J} + \frac{1-\lambda}{1+c},$$

where $\lambda \in [0,1]$ is the weighting factor (a large λ gives a highly accurate model but requires a large number of rules), and c is the maximum number of rules. When the fitness function is evaluated over an empty set, it is undefined. But in this case, one may introduce a penalty factor, $0 < p \ll 1$, and then compute $p\,f(J)$ instead of $f(J)$. If an individual with very high fitness value appears at the ealier stage, this fitness function may cause early convergence of the solution, thereby stopping the algorithm before optimality is reached. To avoid this situation, the individuals are sorted according to their raw fitness values, and the new fitness values are determined recursively by

$$f_1 = 1, \quad f_2 = a\,f_1 = a, \quad \cdots \quad , f_m = a^m,$$

for a fitness scaling factor $a \in (0,1)$. Another example of a fitness function is

$$f_i = \lambda\,\frac{M_i - M_{\min}}{M_{\max} - M_{\min}} + (1-\lambda)\,\frac{R_i - R_{\min}}{R_{\max} - R_{\min}}, \qquad i = 1, \cdots, m,$$

where $\lambda \in [0,1]$ is the weighting factor, M_i is the mean-squared error of the fuzzy model constructed from the ith individual, $M_{\min}$ and $M_{\max}$ are the minimum and maximum mean-squared errors, respectively, R_i is the number of rules of the fuzzy model constructed from the ith individual, $R_{\min}$ and $R_{\max}$ are the minimum and maximum numbers of rules in the generation, respectively, and m is the size of the population.

GA-based fuzzy modeling with fine tuning

As an optimization technique, GA proves to be flexible and capable of identifying simultaneously the parameters in the IF and THEN parts in the rule base of a fuzzy model. However, it generally does not guarantee the convergence to a global optimum, just like many other optimization techniques. In order to improve this situation, the gradient descent method can be used to fine tune the parameters that are identified by GA. Since GA usually can find a near global optimum, to this end a fine tuning of the membership function parameters, in both the IF and THEN parts by a gradient descent method, can generally lead to a global optimization [3].

Let (24) be the objective function to be minimized. Suppose that a simple fuzzy model is considered, whose rule base is

$$R^{(i)}: \quad \text{IF} \;\; x_1 \;\; \text{is} \;\; X_{i1} \;\; \text{AND} \;\; \cdots \;\; \text{AND} \;\; x_{in} \;\; \text{is} \;\; X_{in} \;\; \text{THEN} \;\; y = y_i^o \,,$$

where y_i^o is a real number, $i = 1, \cdots, r$, and whose membership functions are simple triangular functions defined on X_{ij}, with mean point (center point) a_{ij} and width b_{ij}, respectively, $i = 1, \cdots, r$, $j = 1, \cdots, n$.

Given the input data, the numerical output of the fuzzy inference is produced by the weighted average defuzzifier defined in (2), namely,

$$y = \sum_{j=1}^{r} \frac{w_j}{\sum_{j=1}^{r} w_j} \; y_j^o \,,$$

where

$$w_j = \prod_{\ell=1}^{m} \mu_{X_{j\ell}}(x_\ell) \,, \qquad j = 1, \cdots, r \,, \qquad \sum_{i=1}^{r} \frac{w_i}{\sum_{j=1}^{r} w_j} = 1 \,.$$

Learning rules by the gradient descent method can be easily derived via the chain rule, as

$$\begin{cases} a_{ij}(k+1) = a_{ij}(k) - \gamma_a \frac{\partial J}{\partial a_{ij}} \\ b_{ij}(k+1) = b_{ij}(k) - \gamma_b \frac{\partial J}{\partial b_{ij}} \\ y_i^o(k+1) = y_i(k) - \gamma_y \frac{\partial J}{\partial y_i^o} \,, \end{cases} \tag{25}$$

where $\gamma_a, \gamma_b, \gamma_y$ are the learning rates, and

$$\begin{aligned} \frac{\partial J}{\partial a_{ij}} &= \frac{w_i}{\sum_{j=1}^{r} w_j} \left(y_i - \tilde{y}_i\right) \left(y_i^o - \tilde{y}_i\right) \operatorname{sgn}\left(x_j - a_{ij}\right) \frac{2}{b_{ij} \mu_{X_{ij}}(x_j)} \\ \frac{\partial J}{\partial b_{ij}} &= \frac{w_i}{\sum_{j=1}^{r} w_j} \left(y_i - \tilde{y}_i\right) \left(y_i^o - \tilde{y}_i\right) \operatorname{sgn}\left(x_j - a_{ij}\right) \frac{1 - \mu_{X_{ij}}(x_j)}{b_{ij} \mu_{X_{ij}}(x_j)} \\ \frac{\partial J}{\partial y_i} &= \frac{w_i}{\sum_{j=1}^{r} w_j} \left(y_i - \tilde{y}_i\right) . \end{aligned}$$

The computational algorithm of GA-based fuzzy modeling procedure using fine tuning is summarized as follows [11]:

Step 1: Set parameter values for GA: maximum number of generation, population size, crossover rate, mutation rate, maximum number of rules

Step 2: Decode the chromosome of each individual in the population and determine the fuzzy model. Evaluate the determined fuzzy model and assign a fitness value to each individual.

Step 3: Evolve the population by reproduction, crossover and mutation. Increase the generation number by replacing old generation with new generation. During the replacement, preserve the population that has the maximum fitness value by the elitist reproduction.

Step 4: Repeat Steps 2 and 3 until satisfactory population is obtained, or the maximum number of generation is reached.

Step 5: After coarse tuning by GA, the parameters identified by GA are used as initial values for fine tuning.

Step 6: Identify the parameters of the fuzzy model by the gradient descent fine-tuning scheme such that the modeling error is minimized.

Step 7: Repeat Step 6 until the error is less than a predefined value, or the maximum iteration number is reached.

This method of GA-based fuzzy modeling with fine-tuning adjustment is illustrated by an example here for clarification.

An example

Consider a nonlinear system with two inputs and one output [16]:

$$y = \left(1 + x_1^{-2} + x_2^{-1.5}\right)^2, \qquad 1 \le x_1, , x_2 \le 5\,. \tag{26}$$

Suppose that this system is unknown to the designer. The 50 input-output data pairs given in [16] are used for fuzzy modeling here. These input-output data are first normalized to be within $[0, 1]$: for the ℓth data pair,

$$\bar{x}_{1\ell} = \frac{x_{1\ell} - x_{1,\min}}{x_{1,\max} - x_{1,\min}}\,, \quad \bar{x}_{2\ell} = \frac{x_{2\ell} - x_{2,\min}}{x_{2,\max} - x_{2,\min}}\,, \quad \bar{y}_\ell = \frac{y_\ell - y_{\min}}{y_{\max} - y_{\min}}\,.$$

According to these data set, used in [16], $x_{1,\min} = x_{2,\min} = 1.0$, $x_{1,\max} = x_{2,\max} = 5.0$, $y_{\min} = 1.2756$, and $y_{\max} = 6.4488$.

Then, in initial candidate strings, the centers of membership functions are randomly selected from the extended input range $[-0.2, 1.2]$. The initial widths of such membership

functions are randomly selected from interval $[0, w]$, which properly divides the input value range with

$$w = \frac{2(x^{upp} - x^{low})}{n_{ini}},$$

where x^{upp} and x^{low} are the upper and lower bounds of all the system inputs, and n_{ini} is the initial number of fuzzy rules. In this example, $x^{upp} = 1$, $x^{low} = 0$, and $n_{ini} = c$ (the maximum allowable number of rules).

Parameters for running the GA modeling scheme in this example are shown in Tables 4 and 5. Figure 21 shows the mean-squared modeling errors during the GA evolution and the fine-tuning process, respectively. The fuzzy model identified by the GA scheme with fine tuning is described by Table 6, which yields a very small total mean-squared error: 0.05232.

Table 4: Basic Parameters of the GA Scheme

Parameter	Symbol	Value
Maximum number of generation	G_m	10,000
Population size	G	100
Crossover probability	P_c	0.8
Mutation probability	P_m	0.3
Penalty factor	P	0.0001
Weighting factor	λ	0.01
Fitness scaling factor	a	0.7

Table 5: Basic Parameters of the Fine Tuning Process

Parameter	Symbol	Value
Learning rate	γ_a	10^{-6}
Learning rate	γ_b	10^{-6}
Learning rate	γ_y	10^{-3}
Number of iteration		300

(a) GA modeling process

(b) Fine-tuning process

Figure 21: The mean-squared modeling errors.

Table 6: Resulting Fuzzy Rules Identified by GA with Fine Tuning

Rule	x_1		x_2		y
	a_{i1}	b_{i1}	a_{i2}	b_{i2}	
1	-0.03646	0.90357	0.48039	1.09565	0.58749
2	0.54291	1.06199	0.47824	1.09565	0.05328
3	0.21145	0.93724	0.62741	1.06199	0.00618
4	0.54291	1.06199	0.53525	1.04099	-0.01400
5	0.53371	0.93565	-0.05728	0.91551	0.36194

References

[1] A. Abe and M. S. Lan, "Fuzzy rules extraction directly from numerical data for function approximation," *IEEE Trans. on Sys. Man, and Cybern.*, Vol. 25, pp. 119-129, 1995.

[2] B. Carse, T. C. Fofarty, and A. Munro, "Evolving fuzzy rule based controllers using genetic algorithms," *Fuzzy Sets Sys.*, Vol. 80, pp. 273-293, 1996.

[3] W. Chang, Y. H. Joo, J. B. Park and G. Chen, "Robust fuzzy-model-based controller design via genetic algorithms for a class of complex single-input single-output nonlinear systems," preprint, 1998.

[4] G. Chen and T. T. Pham, Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems, CRC Press, 1999.

[5] G. Chen, T. T. Pham, and J. J. Weiss, "Fuzzy modeling of control systems," *IEEE Trans. on Aero. Elect. Sys.*, Vol. 30, pp. 414-429, 1995.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[7] G. C. Goodwin and K. S. Sin, *Adaptive Filtering, Prediction and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[8] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, 1975.

[9] A. Homaifar and V. E. McCormick, "Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms," *IEEE Trans. on Fuzzy Sys.*, Vol. 3, pp. 129-139, 1995.

[10] Y. C. Hsu and G. Chen, "Fuzzy dynamical modeling techniques for nonlinear control systems and their applications to multiple-input multiple-output (MIMO) systems," in *Fuzzy Theory, Systems, Techniques and Applications*, C. T. Leondes (ed.), Academic Press, New York, 1999, in press.

[11] Y. H. Joo, H. S. Hwang, K.B. Kim and K.B. Woo, "Fuzzy system modeling by fuzzy partition and GA hybrid schemes," *Fuzzy Sets Sys.*, Vol. 86, pp. 279-288, 1997.

[12] J. Liska and S. S. Melsheimer, "Complete design of fuzzy logic systems using genetic algorithms," *Proc. of IEEE Conf. on Fuzzy Sys.*, pp. 1377-1382, 1994.

[13] K. F. Man, K. S. Tang, S. Kwong, and W. A. Halang, *Genetic Algorithms for Control and Signal Processing*, Springer, New York, 1997.

[14] B. Soucek and T. I. Group, *Dynamic Genetic and Chaotic Programming*, John Wiley & Sons, Inc., 1992.

[15] W. Spears and V. Anand, "The use of crossover in genetic programming," NRL Tech. Rep., AI Center, Naval Research Labs, Washington D. C., 1990.

[16] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. on Sys., Man, and Cybern.*, Vol. 15, pp. 116-132, 1985.

[17] L. X. Wang, Book *Adaptive Fuzzy Control Systems*, Prentice-Hall, Englewood Cliffs, NJ, 199?.

[18] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. on Sys., Man, and Cybern.*, Vol. 22, pp. 1414-1427, 1996.

[19] L. A. Zadeh, "Fuzzy sets," *Information and Control*, Vol. 8, pp. 338-353, 1965.

[20] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. on Sys., Man, and Cybern.*, Vol. 3, pp. 28-44, 1973.

[21] A. M. S. Zalzala and P. J. Fleming, *Genetic Algorithms in Engineering Systems*, IEE Press, UK, 1997.