A First Course in FUZZY and NEURAL CONTROL

A First Course in FUZZY and NEURAL CONTROL

Hung T. Nguyen • Nadipuram R. Prasad
Carol L. Walker • Elbert A. Walker

CHAPMAN & HALL/CRC

A CRC Press Company
Boca Raton London New York Washington, D.C.

Library of Congress Cataloging-in-Publication Data

A first course in fuzzy and neural control / Hung T. Nguyen ... [et al.].
p. cm.
Includes bibliographical references and index.
ISBN 1-58488-244-1
1. Soft computing. 2. Neural networks (Computer science) 3. Fuzzy systems. 4. Control theory. I. Nguyen, Hung T., 1944-

QA76.9.S63 .F57 2002
006.3—dc21 2002031314

No claim to original U.S. Government works
International Standard Book Number 1-58488-244-1
Library of Congress Card Number 2002031314
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

Contents

Preface

Soft computing approaches in decision making have become increasingly popular in many disciplines. This is evident from the vast number of technical papers appearing in journals and conference proceedings in all areas of engineering, manufacturing, sciences, medicine, and business. Soft computing is a rapidly evolving field that combines knowledge, techniques, and methodologies from various sources, using techniques from neural networks, fuzzy set theory, and approximate reasoning, and using optimization methods such as genetic algorithms. The integration of these and other methodologies forms the core of soft computing.

The motivation to adopt *soft computing*, as opposed to *hard computing*, is based strictly on the tolerance for imprecision and the ability to make decisions under uncertainty. Soft computing is goal driven — the methods used in finding a path to a solution do not matter as much as the fact that one is moving toward the goal in a reasonable amount of time at a reasonable cost. While soft computing has applications in a wide variety of fields, we will restrict our discussion primarily to the use of soft computing methods and techniques in control theory.

Over the past several years, courses in fuzzy logic, artificial neural networks, and genetic algorithms have been offered at New Mexico State University when a group of students wanted to use such approaches in their graduate research. These courses were all aimed at meeting the special needs of students in the context of their research objectives. We felt the need to introduce a formal curriculum so students from all disciplines could benefit, and with the establishment of The Rio Grande Institute for Soft Computing at New Mexico State University, we introduced a course entitled "Fundamentals of Soft Computing I" during the spring 2000 semester. This book is an outgrowth of the material developed for that course.

We have a two-fold objective in this text. Our first objective is to emphasize that both fuzzy and neural control technologies are firmly based upon the principles of classical control theory. All of these technologies involve knowledge of the basic characteristics of system response from the viewpoint of stability, and knowledge of the parameters that affect system stability. For example, the concept of state variables is fundamental to the understanding of whether or not a system is controllable and/or observable, and of how key system variables can be monitored and controlled to obtain desired system performance.

To help meet the first objective, we provide the reader a broad flavor of what classical control theory involves, and we present in some depth the mechanics of implementing classical control techniques. It is not our intent to cover classical methods in great detail as much as to provide the reader with a firm understanding of the principles that govern system behavior and control. As an outcome of this presentation, the type of information needed to implement classical control techniques and some of the limitations of classical control techniques should become obvious to the reader.

Our second objective is to present sufficient background in both fuzzy and neural control so that further studies can be pursued in advanced soft computing methodologies. The emphasis in this presentation is to demonstrate the ease with which system control can be achieved in the absence of an analytical mathematical model. The benefits of a model-free methodology in comparison with a model-based methodology for control are made clear. Again, it is our intent to bring to the reader the fundamental mechanics of both fuzzy and neural control technologies and to demonstrate clearly how such methodologies can be implemented for nonlinear system control.

This text, *A First Course in Fuzzy and Neural Control,* is intended to address all the material needed to motivate students towards further studies in soft computing. Our intent is not to overwhelm students with unnecessary material, either from a mathematical or engineering perspective, but to provide balance between the mathematics and engineering aspects of fuzzy and neural network-based approaches. In fact, we strongly recommend that students acquire the mathematical foundations and knowledge of standard control systems before taking a course in soft computing methods.

Chapter 1 provides the fundamental ideas of control theory through simple examples. Our goal is to show the consequences of systems that either do or do not have feedback, and to provide insights into controller design concepts. From these examples it should become clear that systems can be controlled if they exhibit the two properties of *controllability* and *observability.*

Chapter 2 provides a background of classical control methodologies, including state-variable approaches, that form the basis for control systems design. We discuss state-variable and output feedback as the primary motivation for designing controllers via pole-placement for systems that are inherently unstable. We extend these classical control concepts to the design of conventional *Proportional-Integral* (PI), *Proportional-Derivative* (PD), and *Proportional-Integral-Derivative* (PID) controllers. Chapter 2 includes a discussion of stability and classical methods of determining stability of nonlinear systems.

Chapter 3 introduces mathematical notions used in linguistic rule-based control. In this context, several basic examples are discussed that lay the mathematical foundations of fuzzy set theory. We introduce linguistic rules — methods for inferencing based on the mathematical theory of fuzzy sets. This chapter emphasizes the logical aspects of reasoning needed for intelligent control and decision support systems.

In Chapter 4, we present an introduction to fuzzy control, describing the

general methodology of fuzzy control and some of the main approaches. We discuss the design of fuzzy controllers as well as issues of stability in fuzzy control. We give examples illustrating the solution of control problems using fuzzy logic.

Chapter 5 discusses the fundamentals of artificial neural networks that are used in control systems. In this chapter, we briefly discuss the motivation for neural networks and the potential impact on control system performance. In this context, several basic examples are discussed that lay the mathematical foundations of artificial neural networks. Basic neural network architectures, including single- and multi-layer perceptrons, are discussed. Again, while our objective is to introduce some basic techniques in soft computing, we focus more on the rationale for the use of neural networks rather than providing an exhaustive survey and list of architectures.

In Chapter 6, we lay down the essentials of neural control and demonstrate how to use neural networks in control applications. Through examples, we provide a step-by-step approach for neural network-based control systems design.

In Chapter 7, we discuss the hybridization of fuzzy logic-based approaches with neural network-based approaches to achieve robust control. Several examples provide the basis for discussion. The main approach is adaptive neuro-fuzzy inference systems (ANFIS).

Chapter 8 presents several examples of fuzzy controllers, neural network controllers, and hybrid fuzzy-neural network controllers in industrial applications. We demonstrate the design procedure in a step-by-step manner. Chapters 1 through 8 can easily be covered in one semester. We recommend that a minimum of two projects be assigned during the semester, one in fuzzy control and one in neural or neuro-fuzzy control.

Throughout this book, the significance of simulation is emphasized. We strongly urge the reader to become familiar with an appropriate computing environment for such simulations. In this book, we present MATLAB® simulation models in many examples to help in the design, simulation, and analysis of control system performance. MATLAB can be utilized interactively to design and test prototype controllers. The related program, Simulink®, provides a convenient means for simulating the dynamic behavior of control systems.

We thank the students in the Spring 2000 class whose enthusiastic responses encouraged us to complete this text. We give special thanks to Murali Siddaiah and Habib Gassoumi, former Ph.D. students of Ram Prasad, who kindly permitted us to share with you results from their dissertations that occur as examples in Chapters 6 and 8. We thank Chin-Teng Lin and C. S. George Lee who gave us permission to use a system identification example from their book *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems.*

Much of the material discussed in this text was prepared while Ram Prasad spent a year at the NASA/Jet Propulsion Laboratory between August 2001 and August 2002, as a NASA Faculty Fellow. For this, he is extremely thankful to Anil Thakoor of the Bio-Inspired Technologies and Systems Group, for his constant support, encouragement, and the freedom given to explore both

application-oriented technologies and revolutionary new technology development.

We thank our editor, Bob Stern, project coordinator, Jamie B. Sigal, and project editor Marsha Hecht for their assistance and encouragement. And we give a very personal thank you to Ai Khuyen Prasad for her utmost patience and good will.

Hung T. Nguyen
Nadipuram R. Prasad
Carol L. Walker
Elbert A. Walker

Las Cruces, New Mexico
July 2002

Chapter 1

A PRELUDE TO CONTROL THEORY

In this opening chapter, we present fundamental ideas of control theory through simple examples. These fundamental ideas apply no matter what mathematical or engineering techniques are employed to solve the control problem. The examples clearly identify the concepts underlying open-loop and closed-loop control systems. The need for feedback is recognized as an important component in controlling or regulating system performance. In the next chapter, we will present examples of classical modern control theory systems that rely on *mathematical models*, and in the remainder of this book, we explore possible alternatives to a rigid mathematical model approach. These alternative approaches — fuzzy, neural, and combinations of these — provide alternative designs for autonomous intelligent control systems.

1.1 An ancient control system

Although modern control theory relies on mathematical models for its implementation, control systems were invented long before mathematical tools were available for developing such models. An amazing control system invented about 2000 years ago by Hero of Alexandria, a device for the opening and closing of temple doors — is still viewed as a control system marvel. Figure 1.1 illustrates the basic idea of his vision. The device was actuated whenever the ruler and his entourage arrived to ascend the temple steps. The actuation consisted of lighting a fire upon a sealed altar enclosing a column of air. As the air temperature in the sealed altar increased, the expanding hot air created airflow from the altar into a sealed vessel directly below. The increase in air pressure created inside the vessel pushed out the water contained in this vessel. This water was collected in a bucket. As the bucket became heavier, it descended and turned the door spindles by means of ropes, causing the counterweights to rise. The left spindle rotated in the clockwise direction and the right spindle in the counter-

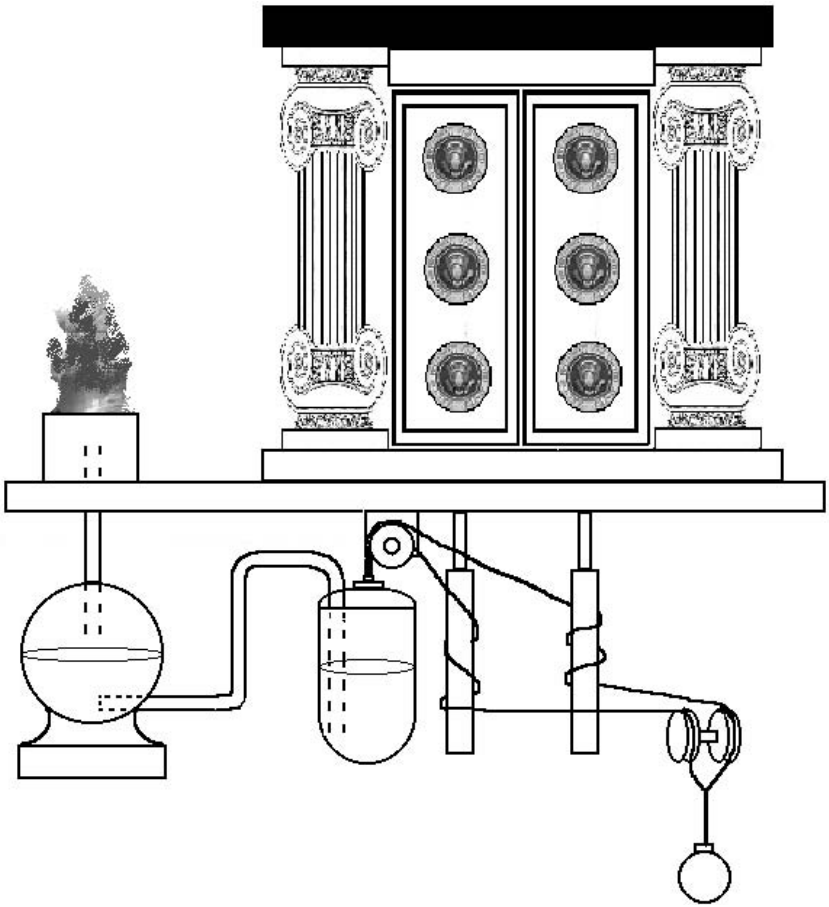


Figure 1.1. Hero's automatic temple doors

clockwise direction, thus opening the temple doors. The bucket, being heavier than the counterweight, would keep the temple doors open as long as the fire upon the altar was kept burning. Dousing the fire with cold water caused the temple doors to close.[1] As the air in the altar cooled, the contracting cool air in the altar created a suction to extract hot air from the sealed vessel. The resulting pressure drop caused the water from the bucket to be siphoned back into the sealed vessel. Thus, the bucket became lighter, and the counterweight

[1] Here, there is a question on how slow or how fast the temple doors closed after dousing out the fire. This is an important consideration, and a knowledge of the exponential decay in temperature of the air column inside the altar holds the answer. Naturally then, to give a theatrical appearance, Hero could have had copper tubes that carried the air column in close contact with the heating and cooling surface. This would make the temperature rise quickly at the time of opening the doors and drop quickly when closing the doors.

being heavier, moved down, thereby closing the door. This system was kept in total secret, thus creating a mystic environment of superiority and power of the Olympian Gods and contributing to the success of the Greek Empire.

1.2 Examples of control problems

One goal of classical science is to understand the behavior of motion of physical systems. In control theory, rather than just to understand such behavior, the object is to force a system to behave the way we want. Control is, roughly speaking, a means to force desired behaviors. The term **control**, as used here, refers generally to an instrument (possibly a human operator) or a set of instruments used to operate, regulate, or guide a machine or vehicle or some other system. The device that executes the control function is called the **controller**, and the system for which some property is to be controlled is called the **plant**. By a **control system** we mean the plant and the controller, together with the

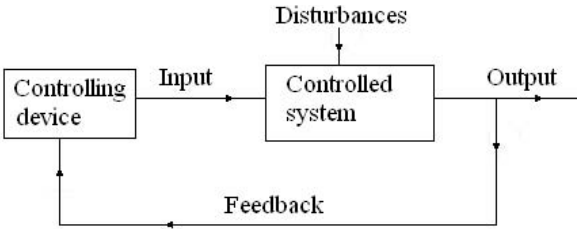


Figure 1.2. Control system

communication between them. The examples in this section include manual and automatic control systems and combinations of these. Figure 1.2 illustrates the basic components of a typical control system. The controlling device produces the necessary input to the controlled system. The output of the controlled system, in the presence of unknown disturbances acting on the plant, acts as a feedback for the controlling device to generate the appropriate input.

1.2.1 Open-loop control systems

Consider a system that is driven by a human — a car or a bicycle for example. If the human did not make observations of the environment, then it would be impossible for the "system" to be controlled or driven in a safe and secure manner. Failure to observe the motion or movement of the system could have catastrophic results. Stated alternatively, if there is no feedback regarding the system's behavior, then the performance of the system is governed by how well the operator can maneuver the system without making any observations of the behavior of the system. Control systems operating without feedback regarding the system's behavior are known as **open-loop control systems**. In other

words, an open-loop control system is one where the control inputs are chosen without regard to the actual system outputs. The performance of such systems can only be guaranteed if the task remains the same for all time and can be duplicated repeatedly by a specific set of inputs.

Example 1.1 (Traffic light) To control the flow of traffic on city streets, a traffic engineer may preset a fixed time interval for a traffic light to turn green, yellow, and red. In this example, the environment around the street intersection is the plant. Traffic engineers are interested in controlling some specified plant

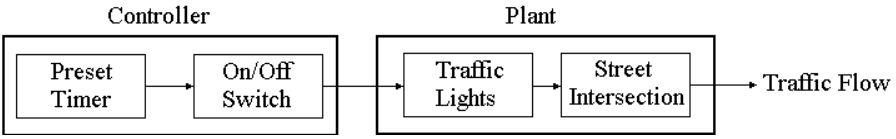


Figure 1.3. Traffic light, open-loop control

output, here the traffic flow. The preset timer and on/off switch for the traffic light comprise the controller. Since the traffic lights operate according to a preset interval of time, without taking into account the plant output (the timing is unaltered regardless of the traffic flow), this control system is an open-loop control system. A pictorial representation of the control design, called a ***block diagram***, is shown in Figure 1.3.

Example 1.2 (Toaster) A toaster can be set for producing the desired darkness of toasted bread. The "darkness" setting allows a timer to time out and switch off the power to the heating coils. The toaster is the plant, and the

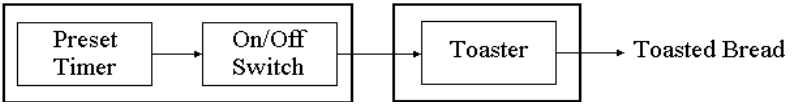


Figure 1.4. Standard toaster

timing mechanism is the controller. The toaster by itself is unable to determine the darkness of the toasted bread in order to adjust automatically the length of time that the coils are energized. Since the darkness of the toasted bread does not have any influence on the length of time heat is applied, there is no feedback in such a system. This system, illustrated in Figure 1.4, is therefore an open-loop control system.

Example 1.3 (Automatic sprinkler system) An automatic home sprinkler system is operated by presetting the times at which the sprinkler turns on and off. The sprinkler system is the plant, and the automatic timer is the controller.

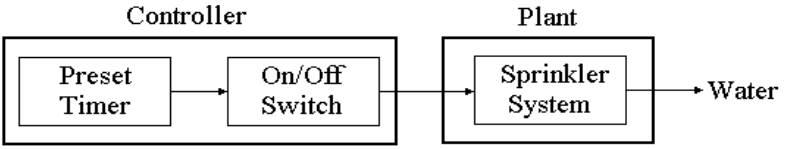


Figure 1.5. Automatic sprinkler system

There is no automatic feedback that allows the sprinkler system to modify the timed sequence based on whether it is raining, or if the soil is dry or too wet. The block diagram in Figure 1.5 illustrates an open-loop control system.

Example 1.4 (Conventional oven) With most conventional ovens, the cooking time is prescribed by a human. Here, the oven is the plant and the controller is the thermostat. By itself, the oven does not have any knowledge of the food

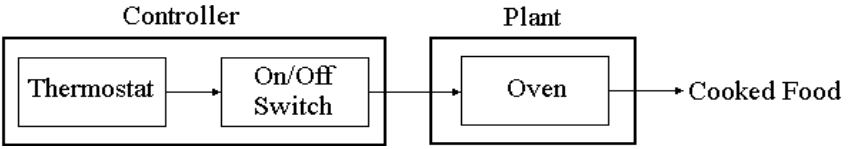


Figure 1.6. Conventional oven

condition, so it does not shut itself off when the food is done. This is, therefore, an open-loop control system. Without human interaction the food would most definitely become inedible. This is typical of the outcome of almost all open-loop control problems.

From the examples discussed in this section, it should become clear that some feedback is necessary in order for controllers to determine the amount of correction, if any, needed to achieve a desired outcome. In the case of the toaster, for example, if an observation was made regarding the degree of darkness of the toasted bread, then the timer could be adjusted so that the desired darkness could be obtained. Similar observations can be made regarding the performance of the controller in the other examples discussed.

1.2.2 Closed-loop control systems

Closed-loop systems, or **feedback control systems**, are systems where the behavior of the system is observed by some sensory device, and the observations are fed back so that a comparison can be made about how well the system is behaving in relation to some desired performance. Such comparisons of the performance allow the system to be controlled or maneuvered to the desired final state. The fundamental objective in closed-loop systems is to make the actual response of a system equal to the desired response.

Example 1.5 (Traffic light) To control the traffic flow in a more efficient

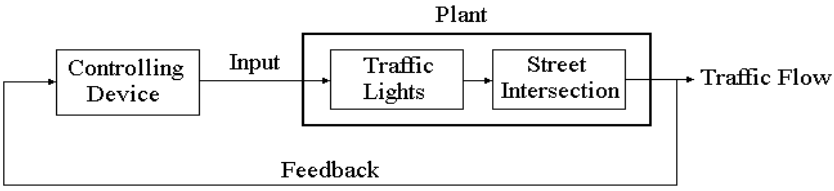


Figure 1.7. Traffic light feedback control

manner than in the example of the open-loop traffic light control described in Example 1.1, we could design a controller that does take into account the traffic flow (i.e., plant output). In this case, the new control system is referred to as a closed-loop system since the control strategy uses feedback information. The block diagram of this design is shown in Figure 1.7.

Example 1.6 (Flush tank) Suppose water flows into a flush tank through a supply valve, and the goal is to keep the water in the tank at a given level.

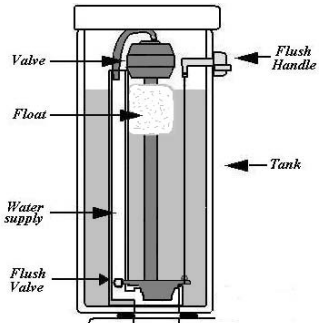


Figure 1.8. (a) Flush tank with float

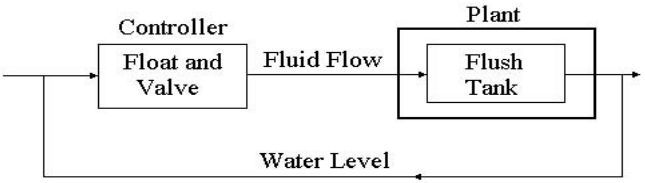


Figure 1.8. (b) Control system diagram for flush tank with float

One control system solving this problem uses a float that opens and closes the supply valve. As the water in the tank rises, the float rises and slowly begins to close the supply valve. When the water reaches the preset level, the supply valve closes shut completely. In this example, the float acts as the observer that

provides feedback regarding the water level. This feedback is compared with the desired level, which is the final position of the float (see Figures 1.8 (a) and (b)).

Example 1.7 (Fluid level) Consider a manually controlled closed-loop system for regulating the level of fluid in a tank (see Figures 1.9 (a) and 1.9 (b)).

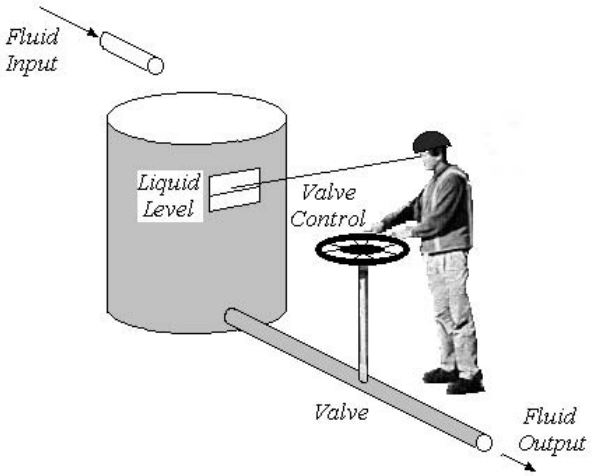


Figure 1.9. (a) Human maintaining fluid level

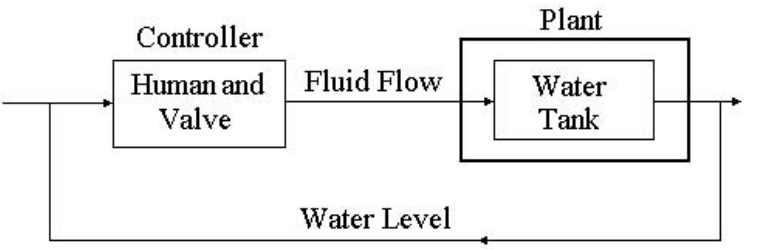


Figure 1.9. (b) Diagram of control system for maintaining fluid level

Fluid input is provided to the tank from a source that you can assume is continuous-time and time-varying. This means that the flow rate of fluid input can change with time. The fluid enters a tank in which there is an outlet for fluid output. The outlet is controlled by a valve, that can be opened or closed to control the flow rate of fluid output. The objective in this control scheme is to maintain a desired level of fluid in the tank by opening or closing the valve controlling the output. Such opening and closing operations either increase or decrease the fluid output flow rate to compensate for variations in the fluid input flow rate.

The operator is instructed to maintain the level of fluid in the tank at a particular level. A porthole on the side of the tank provides the operator a window to observe the fluid level. A reference marker is placed in the window for the operator to see exactly where the fluid level should be. If the fluid level drops below the reference marker, the human sees the fluid level and compares it with the reference. Sensing whether the height of fluid is above or below the reference, the operator can turn the valve either in the clockwise (close) or counterclockwise (open) direction and control the flow rate of the fluid output from the tank.

Good feedback control action can be achieved if the operator can continuously adjust the valve position. This will ensure that the error between the reference marker position, and the actual height of fluid in the tank, is kept to a minimum. The controller in this example is a human operator together with the valve system. As a component of the feedback system, the human operator is performing two tasks, namely, sensing the actual height of fluid and comparing the reference with the actual fluid height. Feedback comes from the visual sensing of the actual position of the fluid in the tank.

Example 1.8 (Temperature control) The temperature inside a home is influenced by the outside temperature. In order to maintain the inside temperature at a comfortable level, the desired room temperature is set on the thermostat. If the room temperature is lower than the desired temperature, a relay

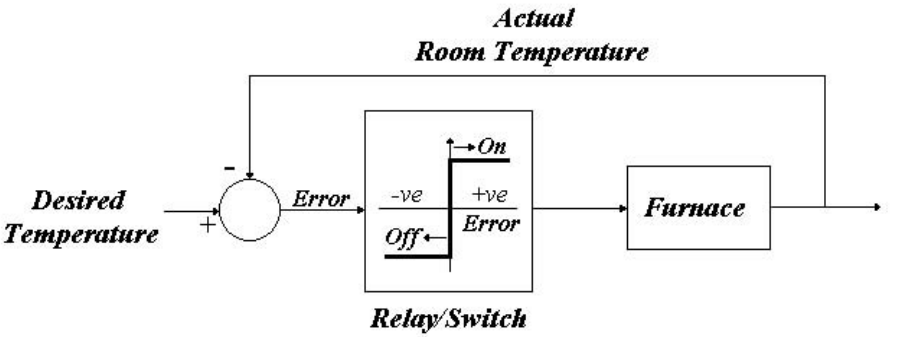


Figure 1.10. Thermostat controlling room temperature

closes and turns on the furnace to produce heat in the room. When the room temperature reaches the desired temperature, the relay opens, and in turn shuts off the furnace.

As shown in Figure 1.10, a comparator is used to determine whether or not the actual room temperature is equal to the desired room temperature. The relay/switch and furnace are the dynamic elements of this closed-loop control system shown in the figure.

1.3 Stable and unstable systems

Stability of an uncontrolled system indicates resistance to change, deterioration, or displacement, in particular the ability of the system to maintain equilibrium or resume its original position after displacement. Any system that violates these characteristics is unstable. A closed-loop system must, aside from meeting performance criteria, be stable.

Example 1.9 (Stable system) The pendulum is among the most stable of all systems. No matter what position the ball is placed in, the pendulum tends toward the vertical "at rest" position, shown in Figure 1.11.

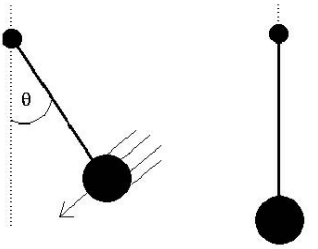


Figure 1.11. Pendulum in motion and at rest

Pendulum clocks have been used to keep time since 1656, and they have not changed dramatically since then. They were the first clocks having a high level of accuracy, made possible by the fact that the period of a pendulum's swing is related only to the length of the pendulum and the force of gravity. When

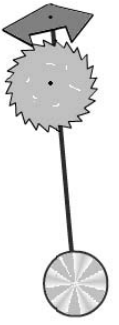


Figure 1.12. Clock's escapement with pendulum

you "wind" a weight-driven clock, you pull on a cord that lifts the weights. The weights act as an energy storage device so that the clock can run unattended for relatively long periods of time. There are also gears that make the minute and hour hands turn at the proper rates. Figure 1.12 shows an escapement with a

gear having teeth of a special shape. Attached to the pendulum is a device to engage the teeth of the gear. For each swing of the pendulum, one tooth of the gear is allowed to "escape." That is what produces the ticking sound of a clock. One additional job of the escapement gear is to impart just enough energy into the pendulum to overcome friction and allow it to keep swinging.

Example 1.10 (Unstable system) An **inverted pendulum** is an upright pole with its fulcrum at the base. The objective is to balance the pole in the upright position by applying the appropriate force at the base. An inverted pendulum is inherently unstable, as you can observe by trying to balance a pole upright in your hand (Figure 1.13). Feedback control can be used to stabilize an inverted pendulum. We will give several examples in later chapters.

Figure 1.13. Balancing inverted pendulum

1.4 A look at controller design

Synthesizing the above examples of control problems, we can describe a typical control problem as follows. For a given plant P, it is desirable to control a

specific plant output y by manipulating a plant input u in such a way to achieve some *control objective*. That is to say, build a device C called a *controller* that will send control signals u to the plant (u is the input to the plant) in such a way as to achieve the given control objective (y is the output from the plant). The function u is referred to as a **control law,** the specification of the control

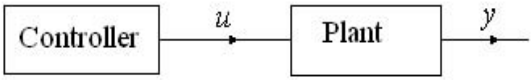


Figure 1.14. Control law

signal. Figure 1.14 illustrates the problem. A *successful control law* is one that does the job. Depending upon whether feedback information is used or not, we have feedback or nonfeedback control laws. The *engineering problem* is this. How do you find the function u and how do you implement it?

Example 1.11 (Cruise control) Suppose we want to keep the speed of a car at $y_0 = 65$ mph for all $t > t_0$. This is an example of a **set-point** control problem. We have at our disposal a force $u(t)$, and we can observe the speed $y(t)$. We consider the open-loop case. By the nature of the control problem, there is a relationship between the input u and the output y, that is, there is a function f satisfying

$$y(t) = f(u(t))$$

Given y_0, the problem is to find the control function $u_0(t)$ such that $f(u_0(t)) = y_0$ for $t > t_0$. It seems obvious that, without knowing f, there is no hope of finding u_0. The function f is referred to as a mathematical model for the plant.

From this viewpoint, standard control theory immediately focuses on finding suitable mathematical models for a given plant as a very first task in the *analysis* and *synthesis* of any control problem. Note that **analysis** means collecting information pertinent to the control problem at hand; whereas **synthesis** means actually constructing a *successful control law.* In most cases, a major part of the effort is devoted to the task of developing a mathematical model for a plant. In general, this is extremely difficult. The task requires detailed knowledge of the plant and knowledge of physical laws that govern the interaction of all the variables within the plant. The model is, at best, an approximate representation of the actual physical system. So, the natural question that arises is whether you can control the plant without knowing the relationship f between u and y — that is, by using a model-free approach.

For our car example, it is straightforward to obtain a mathematical model. From physical laws, the equation of motion (the plant dynamics) is of the form

$$\frac{d^2x(t)}{dt^2} + \frac{adx(t)}{dt} = bu(t) \tag{1.1}$$

where $x(t)$ denotes the car's position.

The velocity is described by the equation $y(t) = dx(t)/dt$, so Equation 1.1, written in terms of y, is

$$\frac{dy(t)}{dt} + ay(t) = bu(t) \tag{1.2}$$

This equation gives rise to the needed relation between the input $u(t)$ and output $y(t)$, namely $y(t) = f(u(t))$. This is done by solving for $u(t)$ for a given $y(t)$. This equation itself provides the control law immediately. Indeed, from it you see that, in order for $y(t) = y_0$, for $t > 0$, the acceleration $dy(t)/dt$ should be equal to zero, so it is sufficient to take $u(t) = (a/b)y_0$ for all $t > 0$.

To solve the second-order linear differential equation in Equation 1.2, you can use Laplace transforms. This yields the **transfer function** $F(s)$ of the plant and puts you in the **frequency domain** — that is, you are working with functions of the complex frequency s. Taking inverse Laplace transforms returns $u(t)$, putting you back in the **time domain**. These transformations often simplify the mathematics involved and also expose significant components of the equations. You will see some examples of this in Chapter 2. Note that this example is not realistic for implementation, but it does illustrate the standard control approach.

The point is that to obtain a control law analytically, you need a mathematical model for the plant. This might imply that if you don't have a mathematical model for your plant, you cannot find a control law analytically. So, how can you control complicated systems whose plant dynamics are difficult to know? A mathematical model may not be a necessary prerequisite for obtaining a successful control law. This is precisely the philosophy of the fuzzy and neural approaches to control.

To be precise, typically, as in several of the preceding examples, feedback control is needed for a successful system. These closed-loop controls are closely related to the heuristics of "If...then..." rules. Indeed, if you feed back the plant output $y(t)$ to the controller, then the control $u(t)$ should be such that the error $y(t) - y_0 = e(t)$ goes to zero. So, apparently, the design of the control law $u(t)$ is reduced to another box with input $e(t)$ and output $u(t)$. Thus,

$$u(t) = g(e(t)) = h(y(t), y_0)$$

The problem is to find the function g or to approximate it from observable values of $u(t)$ and $y(t)$. Even though $y(t)$ comes out from the plant, you don't need the plant's mathematical model to be able to observe $y(t)$. Thus, where does the mathematical model of the plant come to play in standard control theory, in the context of feedback control? From a common-sense viewpoint, we can often suggest various obvious functions g. This is done for the so-called *proportional integral derivative* (PID) types of controllers discussed in the next chapter. However, these controllers are not automatically successful controllers. Just knowing the forms of these controllers is not sufficient information to make them successful. Choosing good parameters in these controllers is a difficult design problem, and it is precisely here that the mathematical model is needed.

In the case of linear and time-invariant systems, the mathematical model can be converted to the so-called transfer functions of the plant and of the controller to be designed. As we will see, knowledge of the poles of these transfer functions is necessary for designing state-variable feedback controllers or PID controllers that will perform satisfactorily.

Even for linear and time-invariant plants, the modern view of control is feedback control. From that viewpoint, a control law is a function of the error. Proposing a control law, or approximating it from training data (a curve fitting problem), are obvious ways to proceed. The important point to note is that the possible forms of a control law are not derived from a mathematical model of the plant, but rather from *heuristics*. What the mathematical model does is help in a systematic analysis leading to the choice of good parameters in the proposed control law, in order to achieve desirable control properties. In other words, *with a mathematical model for the plant, there exist systematic ways to design successful controllers.*

In the absence of a mathematical model for the plant, we can always approximate a plausible control law, either from a collection of "If... then..." rules or from training data. When we construct a control law by any approximation procedures, however, we have to obtain a good approximation. There are no parameters, per se, in this approximation approach to designing control laws. There are of course "parameters" in weights of neural networks, or in the membership functions used by fuzzy rules, but they will be adjusted by training samples or trial and error. There is no need for analytical mathematical models in this process. Perhaps that is the crucial point explaining the success of soft computing approaches to control.

Let us examine a little more closely the prerequisite for mathematical models. First, even in the search for a suitable mathematical model for the plant, we can only obtain, in most cases, a mathematical representation that approximates the plant dynamics. Second, from a common sense point of view, any control strategy is really based upon "If... then..." rules. The knowledge of a functional relationship f provides specific "If... then..." rules, often more than needed. The question is: Can we find control laws based solely on "If... then..." rules? If yes, then obviously we can avoid the tremendous task of spending the major part of our effort in finding a mathematical model. Of course, if a suitable mathematical model is readily available, we generally should use it.

Our point of view is that a weaker form of knowledge, namely a collection of "If...then..." rules, might be sufficient for synthesizing control laws. The rationale is simple: we are seeking an approximation to the control law — that is, the relationship between input and output of the controller directly, and not the plant model. We are truly talking about *approximating functions*. The many ways of approximating an unknown function include using training samples (neural networks) and linguistic "If... then..." rules (fuzzy logic).[2]

[2] In both cases, the theoretical foundation is the so-called *universal approximation capability*, based on the Stone-Weierstrass Theorem, leading to "good" models for control laws.

In summary, standard control theory emphasizes the absolute need to have a suitable mathematical model for the plant in order to construct successful control laws. Recognizing that in formulating a control law we might only need weaker knowledge, neural and fuzzy control become useful alternatives in situations where mathematical models of plants are hard to specify.

1.5 Exercises and projects

1. In Hero's ancient control system, identify the controller and the plant. Develop a block diagram and label various plant details.

2. For the examples shown for open-loop systems, how would you modify each system to provide closed-loop control? Explain with the help of block diagrams both open- and closed-loop systems for each example.

3. The Intelligent Vehicle Highway System (IVHS) program for future transportation systems suggests the possibility of using sensors and controllers to slow down vehicles automatically near hospitals, accident locations, and construction zones. If you were to design a system to control the flow of traffic in speed-restricted areas, what are the major considerations you have to consider, knowing that the highway system is the controller and the vehicle is the plant? Draw a block diagram that illustrates your design concept. Explain the workings of the IVHS system design.

4. A moving sidewalk is typically encountered in large international airports. Design a moving sidewalk that operates only when a passenger approaches the sidewalk and stops if there are no passengers on, or approaching, the sidewalk. Discuss what type of sensors might be used to detect the approach of passengers, and the presence of passengers on the sidewalk.

5. A baggage handling system is to be designed for a large international airport. Baggage typically comes off a conveyor and slides onto a carousel that goes around and around. The objective here is to prevent one bag from sliding onto another bag causing a pile up. Your task is to design a system that allows a bag to slide onto the carousel only if there is room between two bags, or if there are no bags. Explain your system with the aid of a block diagram of the control system.

6. A soda bottling plant requires sensors to detect if bottles have the right amount of soda and a metal cap. With the aid of sketches and block diagrams, discuss in detail how you would implement a system of sensors to detect soda level in the bottles and whether or not there is a metal cap on each bottle of soda. State all your assumptions in choosing the type of sensor(s) you wish to use.

7. A potato chip manufacturing plant has to package chips with each bag of chips having a net weight of 16 ounces or 453.6 grams. Discuss in detail how a system can be developed that will guarantee the desired net weight.

Chapter 2

MATHEMATICAL MODELS IN CONTROL

In this chapter we present the basic properties of control and highlight significant design and operating criteria of *model-based* control theory. We discuss these properties in the context of two very popular classical methods of control: state-variable feedback control, and proportional-integral-derivative (PID) control. This chapter serves as a platform for discussing the desirable properties of a control system in the context of fuzzy and neural control in subsequent chapters. It is not our intent to present a thorough treatment of classical control theory, but rather, to present relevant material that provides the foundations for fuzzy and neural control systems. The reader therefore is urged to refer to the many excellent sources in classical control theory for further information.

Standard control theory consists of two tasks, analysis and synthesis. **Analysis** refers to the study of the plant and the control objectives. **Synthesis** refers to designing and building the controller to achieve the objectives. In standard control theory, mathematical models are used in both the analysis and the synthesis of controllers.

2.1 Introductory examples: pendulum problems

We present two simple, but detailed, examples to bring out the general framework and techniques of standard control theory. The first is a simple pendulum, fixed at one end, controlled by a rotary force; and the second is an inverted pendulum with one end on a moving cart. The concepts introduced in these examples are all discussed more formally, and in more detail, later in this chapter.

2.1.1 Example: fixed pendulum

We choose the problem of controlling a pendulum to provide an overview of standard control techniques, following the analysis in [70]. In its simplified

form, the mathematical model of the motion of a pendulum, which is derived from mechanics, is

$$\ddot{\theta}(t) + \sin\theta(t) = u(t) \tag{2.1}$$

where $\theta(t)$ denotes the angle at time t, $\ddot{\theta}(t)$ is the second derivative of $\theta(t)$, k is a constant, and $u(t)$ is the torque applied at time t. See Figure 2.1. Note

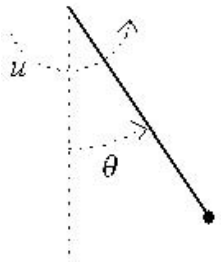


Figure 2.1. Motion of pendulum

that Equation (2.1) is a *nonlinear* differential equation.

The vertical position $\theta = \pi$ is an *equilibrium point* when $\dot{\theta} = 0$ and $u = 0$, but it is unstable. We can make a change of variable to denote this equilibrium point as zero: Let $\varphi = \theta - \pi$, then this equilibrium point is ($\varphi = 0$, $\dot{\varphi} = 0$, $u = 0$).

Suppose we would like to keep the pendulum upright, as shown in Figure 2.2, by manipulating the torque $u(t)$. The appropriate $u(t)$ that does the job

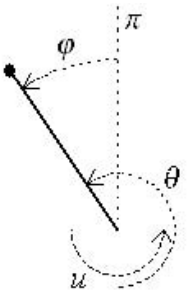


Figure 2.2. Upright pendulum

is called the **control law** of this system. It is clear that in order to achieve our control objective, we need to answer two questions:

1. How do we derive a control law from Equation (2.1)?

2. If such a control law exists, how do we implement it?

In this example, we concentrate on answering the first question. When we attempt to keep the pendulum upright, our operating range is a small range around the unstable equilibrium position. As such, we have a *local* control problem, and we can simplify the mathematical model in Equation (2.1) by linearizing it around the equilibrium point. For $\varphi = \theta - \pi$ small, we keep only the first-order term in the Taylor expansion of $\sin\theta$, that is, $-(\theta - \pi)$, so that the linearization of Equation (2.1) is the linear model (the *linear differential equation*)

$$\ddot{\varphi}(t) - \varphi(t) = u(t) \tag{2.2}$$

and the control objective is manipulating $u(t)$ to bring $\varphi(t)$ and $\dot{\varphi}(t)$ to zero from any small nonzero initial $\varphi(0)$, $\dot{\varphi}(0)$.

Note that Equation (2.2) is a second-order differential equation. It is convenient to replace Equation (2.2) by a system of first-order differential equations in terms of $\varphi(t)$ and $\dot{\varphi}(t)$. Here, let $\mathbf{x}(t)$ be the vector

$$\mathbf{x}(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} = \begin{pmatrix} \varphi(t) \\ \dot{\varphi}(t) \end{pmatrix}$$

so that

$$\dot{\mathbf{x}}(t) = \begin{pmatrix} \dot{\varphi}(t) \\ \ddot{\varphi}(t) \end{pmatrix} = \begin{pmatrix} x_2(t) \\ \dot{x}_2(t) \end{pmatrix}$$

With this notation we see that the original model, Equation (2.1), is written as

$$\dot{\mathbf{x}} = f(\mathbf{x}, u) \tag{2.3}$$

where f is nonlinear, and

$$f = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$

where $f_1(x, u) = x_2$ and $f_2(x, u) = -\sin(x_1 + \pi) + u$. Since f is continuously differentiable and $f(0, 0) = 0$, we can linearize f around $(x, u) = (0, 0)$ as

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu \tag{2.4}$$

where the matrices A and B are

$$\begin{aligned} A &= \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ B &= \begin{pmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

with both Jacobian matrices A and B evaluated at $(x, u) = (0, 0)$.

Thus, in the state-space representation, Equation (2.2) is replaced by Equation (2.4). Note that, in general, systems of the form (2.4) are called **linear systems**, and when A and B do not depend on time, they are called **time-invariant** systems.

Now back to our control problem. Having simplified the original dynamics, Equation (2.1) to the nicer form of Equation (2.2), we are now ready for the analysis leading to the derivation of a control law $u(t)$. The strategy is this. By examining the system under consideration and our control objective, the form of $u(t)$ can be suggested by common sense or naive physics. Then the mathematical model given by Equation (2.2) is used to determine (partially) the control law $u(t)$.

In our control example, $u(t)$ can be suggested from the following heuristic "If .. then ..." rules:

If φ is positive, then u should be negative
If φ is negative, then u should be positive

From these common sense "rules," we can conclude that $u(t)$ should be of the form

$$u(t) = -\alpha\varphi(t) \tag{2.5}$$

for some $\alpha > 0$. A control law of the form (2.5) is called a **proportional control law**, and α is called the **feedback gain**. Note that (2.5) is a feedback control since it is a function of $\varphi(t)$.

To obtain $u(t)$, we need α and $\varphi(t)$. In implementation, with an appropriate gain α, $u(t)$ is determined since $\varphi(t)$ can be measured directly by a sensor. But before that, how do we know that such a control law will stabilize the inverted pendulum? To answer this, we substitute Equation (2.5) into Equation (2.2), resulting in the equation

$$\ddot{\varphi}(t) - \varphi(t) + \alpha\varphi(t) = 0 \tag{2.6}$$

In a sense, this is analogous to guessing the root of an equation and checking whether it is indeed a root. Here, in control context, checking that $u(t)$ is satisfactory or not amounts to checking if the solution $\varphi(t)$ of Equation (2.6) converges to 0 as $t \to +\infty$, that is, checking whether the controller will stabilize the system. This is referred to as the control system (the plant and the controller) being **asymptotically stable**.

For this purpose, we have, at our disposal, the theory of stability of linear differential equations. Thus, we examine the characteristic equation of (2.6), namely

$$z^2 + \alpha - 1 = 0 \tag{2.7}$$

For $\alpha > 1$, the roots of (2.7) are purely imaginary: $z = \pm j\sqrt{\alpha - 1}$, where $j = \sqrt{-1}$. As such, the solutions of (2.6) are all oscillatory and hence do not converge to zero. For $\alpha \leq 1$, it can also be seen that $\varphi(t)$ does not converge to zero as $t \to +\infty$. Thus, $u(t) = -\alpha\varphi(t)$ is not a good guess.

Let us take another guess. By closely examining why the proportional control does not work, we propose to modify it as follows. Only for $\alpha > 1$ do we have hope to modify $u(t)$ successfully. In this case, the torque is applied in the correct direction, but at the same time it creates more inertia, resulting in oscillations of the pendulum. Thus, it appears we need to add to $u(t)$ something that acts

like a brake. In technical terms, we need to add *damping* to the system. The modified control law is now

$$u(t) = -\alpha\varphi(t) - \beta\dot{\varphi}(t) \tag{2.8}$$

for $\alpha > 1$ and $\beta > 0$. Because of the second term in $u(t)$, these types of control laws are called **proportional-derivative** (feedback) control laws, or simply PD control.

To determine if Equation (2.8) is a good guess, as before, we look at the characteristic equation

$$z^2 + \beta z + \alpha - 1 = 0 \tag{2.9}$$

of the closed-loop, second-order linear differential equation

$$\ddot{\varphi}(t) + \beta\dot{\varphi}(t) + (\alpha - 1)\varphi(t) = 0 \tag{2.10}$$

For $\alpha > 1$ and $\beta > 0$, the roots of Equation (2.9) are

$$z = \frac{-\beta \pm \sqrt{\beta^2 - 4(\alpha - 1)}}{2}$$

and hence both have negative real parts. Basic theorems in classical control theory then imply that all solutions of Equation (2.2) with this control law will converge to zero as t gets large. In other words, the PD control laws will do the job. In practice, suitable choices of α and β are needed to implement a good controller. Besides α and β, we need the value $\dot{\varphi}(t)$, in addition to $\varphi(t)$, in order to implement $u(t)$ by Equation (2.8).

Suppose we can only measure $\varphi(t)$ but not $\dot{\varphi}(t)$, that is, our measurement of the state

$$x(t) = \begin{pmatrix} \varphi(t) \\ \dot{\varphi}(t) \end{pmatrix}$$

is of the form

$$y(t) = Cx(t) \tag{2.11}$$

for some known matrix C. Here, $C = \begin{pmatrix} 1 & 0 \end{pmatrix}$.

Equation (2.11) is called the **measurement** (or **output**) **equation**, that, in general, is part of the specification of a control problem (together with (2.4) in the state-space representation).

In a case such as the above, a linear feedback control law that depends only on the allowed measurements is of the form

$$u(t) = KCx(t)$$

Of course, to implement $u(t)$, we need to estimate the components of $x(t)$ that are not directly measured, for example $\dot{\varphi}(t)$, by some procedures. A control law obtained this way is called a **dynamic controller**.

At this point, it should be mentioned that u and y are referred to as **input** and **output**, respectively. Approximating a system from input-output observed data is called **system identification**.

Let us further pursue our problem of controlling an inverted pendulum. The linearized model in Equation (2.2) could be perturbed by some disturbance e, say, resulting in

$$\ddot{\varphi}(t) - \varphi(t) = u(t) + e \tag{2.12}$$

To see if our PD control law is sufficient to handle this new situation, we put (2.8) into (2.12), resulting in

$$\ddot{\varphi}(t) + \beta\dot{\varphi}(t) + (\alpha - 1)\varphi(t) = e \tag{2.13}$$

and examine the behavior of the solutions of (2.13) for t large. It can be shown, unfortunately, that no solutions of (2.13) converge to zero as $t \to +\infty$. So we need to modify (2.8) further to arrive at an acceptable control law. Without going into details here (but see examples in Section 2.7), the additional term to add to our previous PD control law is of the form

$$-\gamma \int_0^t \varphi(s)\, ds$$

This term is used to offset a nonzero error in the PD control. Thus, our new control law takes the form

$$u(t) = -\alpha\varphi(t) - \beta\dot{\varphi}(t) - \gamma \int_0^t \varphi(s)\, ds \tag{2.14}$$

A control law of the form (2.14) is called a **proportional-integral-derivative** (PID) control. PID control is very popular in designing controllers for linear systems. It is important to note that, while PID controls are derived heuristically, stability analysis requires the existence of mathematical models of the dynamics of the systems, and stability analysis is crucial for designing controllers.

In our control example, we started out with a nonlinear system. But since our control objective was local in nature, we were able to linearize the system and then apply powerful techniques in linear systems. For global control problems, as well as for highly nonlinear systems, one should look for nonlinear control methods. In view of the complex behaviors of nonlinear systems, there are no systematic tools and procedures for designing nonlinear control systems. The existing design tools are applicable to particular classes of control problems. However, stability analysis of nonlinear systems can be based on Lyapunov's stability theory.

2.1.2 Example: inverted pendulum on a cart

We look at a standard approach for controlling an inverted pendulum, which we will contrast later with fuzzy control methods. The following mechanical system is referred to as an **inverted pendulum system.** In this system, illustrated in Figure 2.3, a rod is hinged on top of a cart. The cart is free to move in the horizontal plane, and the objective is to balance the rod in the vertical position. Without any control actions on the cart, if the rod were initially in the vertical

position then even the smallest external disturbance on the cart would make the rod lose balance and hence make the system unstable. The objective is to overcome these external perturbations with control action and to keep the rod in the vertical position. Therefore, in the presence of control actions the force on the cart is comprised of both external disturbances and the necessary control actions from a controller to overcome the effects of disturbances.

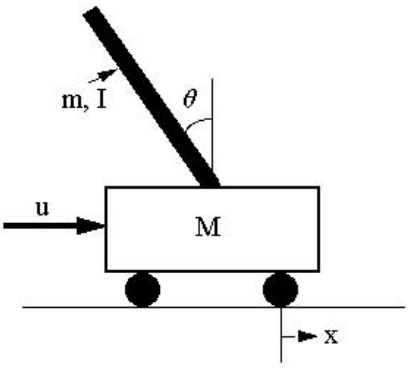


Figure 2.3. Inverted pendulum on a cart

The task of the controller is to apply an appropriate force $u(t)$ to the cart to keep the rod standing upright. We wish to design a controller that can control both the pendulum's angle and the cart's position.

The following model parameters will be used to develop the mathematical model of the system.

M is the mass of the cart.
m is the mass of the pendulum.
b is the friction of the cart resisting motion.
L is the length of the pendulum to its center of mass.
I is the inertia of the pendulum.
$u(t)$ is the force applied to the cart.
x represents the cart position coordinate.
θ is the angle of the pendulum measured from the vertical.

To design a controller for the inverted pendulum from a standard control viewpoint, it is first necessary to determine its mathematical model. In Figure 2.4, we consider the free-body diagrams of the cart and the pendulum. This will allow us to write the equations of motion.

Since the cart can only move around in a horizontal line, we are only interested in obtaining the equation by summing the forces acting on the cart in the horizontal direction. Summing the forces along the horizontal for the cart, we obtain the equation of motion for the cart as

$$M\ddot{x} + b\dot{x} + H = u$$

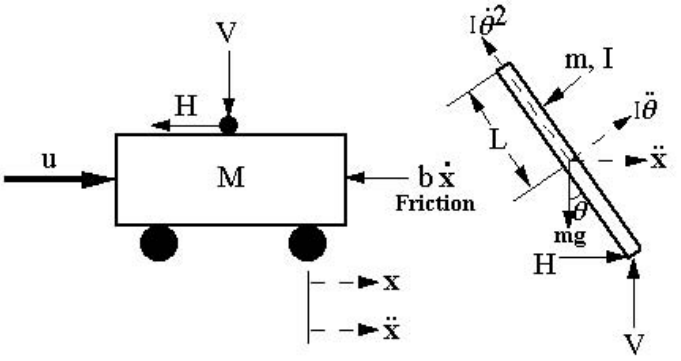


Figure 2.4. Free-body diagrams of the cart and the pendulum

By summing the forces along the horizontal for the pendulum, we get the following equation of motion:

$$H = m\ddot{x} + mL\ddot{\theta}\cos\theta - mL\dot{\theta}^2\sin\theta$$

Substituting this equation into the equation of motion for the cart and collecting terms gives

$$(M+m)\ddot{x} + b\dot{x} + mL\ddot{\theta}\cos\theta - mL\dot{\theta}^2\sin\theta = u \tag{2.15}$$

This is the first of two equations needed for a mathematical model.

The second equation of motion is obtained by summing all the forces in the vertical direction for the pendulum. Note that, as we pointed out earlier, we only need to consider the horizontal motion of the cart; and as such, there is no useful information we can obtain by summing the vertical forces for the cart. By summing all the forces in the vertical direction acting on the pendulum, we obtain

$$V\sin\theta + H\cos\theta - mg\sin\theta = mL\ddot{\theta} + m\ddot{x}\cos\theta$$

In order to eliminate the H and V terms, we sum the moments around the centroid of the pendulum to obtain

$$-VL\sin\theta - HL\cos\theta = I\ddot{\theta}$$

Substituting this in the previous equation and collecting terms yields

$$(mL^2 + I)\ddot{\theta} + mgL\sin\theta = -mL\ddot{x}\cos\theta \tag{2.16}$$

Equations 2.15 and 2.16 are the equations of motion describing the nonlinear behavior of the inverted pendulum. Since our objective is to design a controller for this nonlinear problem, it is necessary for us to linearize this set of equations. Our goal is to linearize the equations for values of θ around π, where $\theta = \pi$ is the vertical position of the pendulum. Consider values of $\theta = \pi + \varphi$ where φ

represents small deviations around the vertical position. For this situation, we can use the approximations $\cos\theta = -1$, $\sin\theta = -\varphi$, and $\ddot{\theta} = 0$. By substituting these approximations into Equations 2.15 and 2.16, we obtain

$$(M+m)\ddot{x} + b\dot{x} - mL\ddot{\varphi} = u \tag{2.17}$$

and

$$(mL^2 + I)\ddot{\varphi} - mgL\varphi = mL\ddot{x} \tag{2.18}$$

Equations 2.17 and 2.18 are the linearized set of equations we will use to design the PID controller.

We first derive the transfer function for the inverted pendulum. To do this, we take the Laplace transform of Equations 2.17 and 2.18 with zero initial conditions which yields

$$(M+m)s^2X(s) + bsX(s) - mLs^2\Phi(s) = U(s) \tag{2.19}$$

and

$$(mL^2 + I)s^2\Phi(s) - mgL\Phi(s) = mLs^2X(s) \tag{2.20}$$

Since we are interested in the deviation $\Phi(s)$ in the pendulum from the vertical position, as a function of the state $X(s)$, we solve Equation 2.20 for $X(s)$ to obtain

$$X(s) = \left[\frac{\left(mL^2 + I\right)}{mL} - \frac{g}{s^2}\right]\Phi(s) \tag{2.21}$$

Substituting Equation 2.21 into 2.19, we obtain the relationship between $\Phi(s)$ and the state $X(s)$ as

$$\begin{aligned}(M+m)\left[\tfrac{\left(mL^2+I\right)}{mL} + \tfrac{g}{s}\right]s^2\Phi(s) + b\left[\tfrac{\left(mL^2+I\right)}{mL} + \tfrac{g}{s}\right]s\Phi(s) - mLs^2\Phi(s) \\ = U(s)\end{aligned} \tag{2.22}$$

Rearranging Equation 2.22, we obtain the transfer function

$$\begin{aligned}\frac{\Phi(s)}{U(s)} &= \frac{\frac{mL}{r}s^2}{s^4 + \frac{b(mL^2+I)}{r}s^3 - \frac{(M+m)mgL}{r}s^2 - \frac{bmgL}{r}s} \\ &= \frac{\frac{mL}{r}s}{s^3 + \frac{b(mL^2+I)}{r}s^2 - \frac{(M+m)mgL}{r}s - \frac{bmgL}{r}}\end{aligned}$$

where

$$r = \left[(M+m)\left(mL^2 + I\right) - (mL)^2\right]$$

Using the method outlined earlier, the linearized equations may be expressed

in state-space form as

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{\varphi}_1(t) \\ \dot{\varphi}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-\left(mL^2+I\right)b}{(M+m)I+MmL^2} & \frac{m^2gL^2}{(M+m)I+MmL^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mLb}{(M+m)I+MmL^2} & \frac{mgL(M+m)}{(M+m)I+MmL^2} & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \varphi_1(t) \\ \varphi_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\left(mL^2+I\right)}{(M+m)I+MmL^2} \\ 0 \\ \frac{mL}{(M+m)I+MmL^2} \end{bmatrix} u\,(t)$$

where $\dot{x}_1(t) = \dot{x}$, $\dot{x}_2(t) = \ddot{x}$, $\dot{\varphi}_1(t) = \dot{\varphi}$ and $\dot{\varphi}_2(t) = \ddot{\varphi}$. Since we are interested in the position of the cart, as well as the angular position of the pendulum, the output may be synthesized as

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \varphi_1(t) \\ \varphi_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u\,(t)$$

For this example we will assume the following parameters:

$$\begin{aligned} M &= 0.5\,\mathrm{kg} \\ m &= 0.2\,\mathrm{kg} \\ b &= 0.1\,\mathrm{N/\,m/\,s} \\ l &= 0.3\,\mathrm{m} \\ I &= 0.006\,\mathrm{kg\,m^2} \end{aligned}$$

To make the design more challenging, we will be applying a step input to the cart. The cart should achieve its desired position within 5 seconds and have a rise time under 0.5 seconds. We will also limit the pendulum's overshoot to 20 degrees (0.35 radians), and it should also settle in under 5 seconds. The design requirements for the inverted pendulum are therefore

- settling time for x and θ of less than 5 seconds,
- rise time for x of less than 0.5 seconds, and
- overshoot of θ less than 20 degrees (0.35 radians).

We use MATLAB to perform several computations. First, we wish to obtain the transfer function for the given set of parameters. Using the *m-file* shown below, we can obtain the coefficients of the numerator and denominator polynomials.

```
M = .5;
m = 0.2;
```

```
b = 0.1;
i = 0.006;
g = 9.8;
l = 0.3;
r = (M+m)*(i+m*l*l)-(m*l)*(m*l);
numplant = [m*l/q 0];
denplant = [1 b*(i+m*l^2)/q -(M+m)*m*g*l/q -b*m*g*l/q];
```

The coefficients of the numerator and denominator polynomial from the MATLAB output can be translated to the following plant transfer function:

$$G_p(s) = \frac{4.5455}{s^3 + 0.1818s^2 - 31.1818s - 4.4545} = \frac{N_p(s)}{D_p(s)}$$

The open-loop response of this transfer function can be simulated in MATLAB using the following code:

```
t = 0:0.01:5;
impulse(numplant,denplant,t)
axis([0 0.9 0 60]);
```

The plot shown in Figure 2.5 clearly indicates the unstable nature of the plant in an open-loop setup.

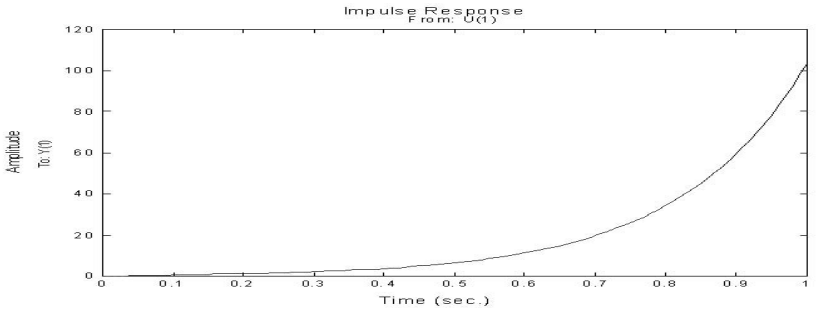


Figure 2.5. Unstable plant

We can now extend the MATLAB script file to include computation of the state-space model. The necessary code is as follows:

```
p = i*(M+m)+M*m*l*l;
A = [0 1 0 0;
0 -(i+m*l*l)*b/p (m*m*g*l*l)/p 0;
0 0 0 1;
0 -(m*l*b)/p m*g*l*(M+m)/p 0]
B = [ 0;
(i+m*l*l)/p;
0;
m*l/p]
```

```
C = [1 0 0 0;
0 0 1 0]
D = [0;
0]
```

The output gives the following state-space model

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{\varphi}_1(t) \\ \dot{\varphi}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.1818 & 2.6727 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -0.4545 & 31.1818 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \varphi_1(t) \\ \varphi_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1.8182 \\ 0 \\ 4.5455 \end{bmatrix} u(t)$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \varphi_1(t) \\ \varphi_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u(t)$$

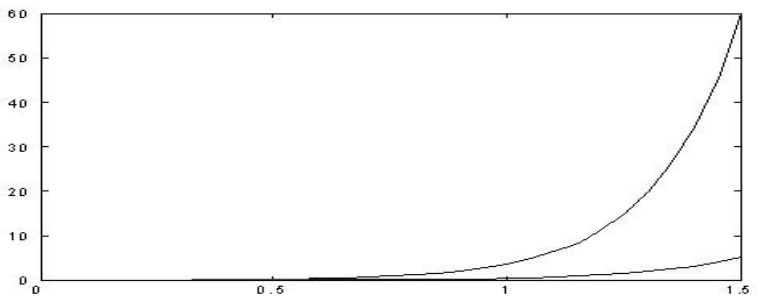


Figure 2.6. Time simulation for a unit step

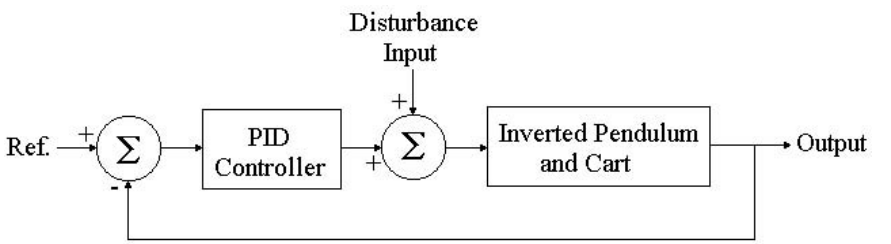


Figure 2.7. Original control structure

Figure 2.6 shows the response of the open-loop system where the system is unstable and Figure 2.7 illustrates the closed-loop control structure for this problem. Note that the control objective is to bring the pendulum to the upright position. As such, the output of the plant is tracking a zero reference with the

vertical reference set to a zero value. Hence, the control structure may be redrawn as shown in Figure 2.8. The force applied to the cart is added as an impulse disturbance.

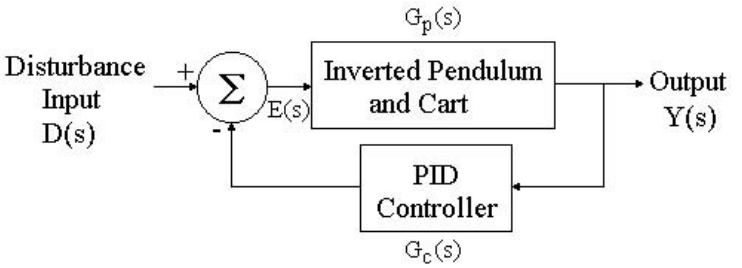


Figure 2.8. Modified control structure

From the modified control structure, we can obtain the closed-loop transfer function that relates the output with the disturbance input. Referring to Figure 2.8,

$$E(s) = D(s) - G_c(s)Y(s)$$

and

$$Y(s) = G_p(s)E(s)$$

Therefore,

$$\frac{Y(s)}{D(s)} = \frac{G_p(s)}{[1 + G_p(s)G_c(s)]}$$

Defining the transfer function of the PID controller

$$G_c(s) = \frac{N_c(s)}{D_c(s)}$$

and using

$$G_p(s) = \frac{N_p(s)}{D_p(s)}$$

we can write the transfer function as

$$\frac{Y(s)}{D(s)} = \frac{\frac{N_p(s)}{D_p(s)}}{[1 + \frac{N_c(s)}{D_c(s)}\frac{N_p(s)}{D_p(s)}]} = \frac{N_p(s)D_c(s)}{[D_c(s)D_p(s) + N_c(s)N_p(s)]}$$

Since the transfer function for the PID controller is

$$G_c(s) = \frac{\left(s^2K_D + sK_P + K_I\right)}{s} = \frac{N_c(s)}{D_c(s)}$$

and the transfer function of the inverted pendulum with a cart is

$$G_p(s) = \frac{4.5455}{s^3 + 0.1818s^2 - 31.1818s - 4.4545} = \frac{N_p(s)}{D_p(s)}$$

we can easily manipulate the transfer function in MATLAB for various numerical values of K_P, K_D, and K_I. To convolve the polynomials in the denominator of the transfer function, we use a special function called *polyadd.m* that is added to the library. The function *polyadd.m* is not in the MATLAB toolbox. This function will add two polynomials even if they do not have the same length. To use *polyadd.m* in MATLAB, enter *polyadd(poly1, poly2)*. Add the following code to your work folder.

```
function[poly]=polyadd(poly1,poly2)
% Copyright 1996 Justin Shriver
% polyadd(poly1,poly2) adds two polynomials possibly of unequal length
if length(poly1)<length(poly2)
  short=poly1;
  long=poly2;
else
  short=poly2;
  long=poly1;
end
mz=length(long)-length(short);
if mz>0
  poly=[zeros(1,mz),short]+long;
else
  poly=long+short;
end
```

It is very convenient to develop controllers using the Simulink features of MATLAB. For the inverted pendulum problem, the simulation diagram using Simulink is shown in Figure 2.9. Parameters for the PID controller can be varied to examine how the system will perform.

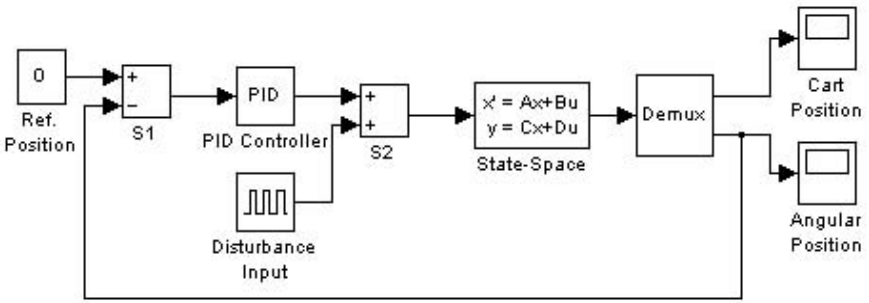


Figure 2.9. Simulink model for inverted pendulum problem

2.2 State variables and linear systems

We now begin to formalize the general framework of standard control, as exemplified by the two previous examples. The state of a system at a given time t is described by a set of variables $x_i(t)$, $i = 1, 2, \ldots, n$, called **state variables**. These variables, that are *functions*, are usually written in the form of a vector function

$$\mathbf{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))$$

The standard **mathematical model** of a control system is a system of differential equations

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t)$$

involving the state variables and the **input** (control) **variables** (also functions)

$$\mathbf{u}(t) = (u_1(t), u_2(t), \ldots, u_k(t))$$

so that the future state of the system can be determined from it. These differential equations are called **state equations**.

In general, state variables cannot be measured directly, but instead, only values of some other set of variables

$$\mathbf{y}(t) = g(\mathbf{x}(t), \mathbf{u}(t)) = (y_1(t), y_2(t), \ldots, y_m(t))$$

called **output variables** can be measured. The equation

$$\mathbf{y} = g(\mathbf{x}, \mathbf{u})$$

is called the **output equation**.

A system whose performance obeys the **principle of superposition** is defined as a **linear system**. The principle states that the mathematical model of a system is **linear** if, when the response to an input $\mathbf{u}$ is $g(\mathbf{x}, \mathbf{u})$, then the response to the linear combination

$$c\mathbf{u} + d\mathbf{v}$$

of inputs is that same linear combination

$$cg(\mathbf{x}, \mathbf{u}) + dg(\mathbf{x}, \mathbf{v})$$

of the corresponding outputs. Here, c and d are constants. The first model of a situation is often constructed to be linear because linear mathematics is very well-developed and methods for control of linear systems are well-understood. In practice, a linear system is an approximation of a nonlinear system near a point, as is explained in Section 2.9. This leads to a piecewise linear system and gives rise to simplified matrix algebra of the form discussed here. Most systems in classical control theory are modeled as piecewise linear systems, and controllers are designed to control the approximated systems.

The system is called **time invariant** if the response to $\mathbf{u}(t-\tau)$ is $\mathbf{y}(t-\tau)$, that is,

$$g(\mathbf{x}(t-\tau), \mathbf{u}(t-\tau)) = \mathbf{y}(t-\tau)$$

for any fixed τ. Linear time invariant sets of state equations are the easiest to manage analytically and numerically. Furthermore, the technique of linearization of nonlinear systems is an important one, relying on the fact that if the perturbation $\mathbf{z}(t)$ from some desired state $\mathbf{x}(t)$ is small, then a set of linear equations in $\mathbf{z}$ can be formed by neglecting all but the first terms in a Taylor series expansion in $\mathbf{z}$.

A **linear differential equation** is one that can be written in the form

$$b_n(x)\, y^{(n)} + b_{n-1}(x)\, y^{(n-1)} + \cdots + b_1(x)\, y' + b_0(x)\, y = R(x) \tag{2.23}$$

where $b_1, ..., b_n$, and R are arbitrary functions of x. Writing D for the differentiation operator $Dy = dy/dx$, and letting a power of D denote repeated differentiation, that is, using the notation

$$D^n y = \frac{d^n y}{dx^n}$$

the left side of Equation 2.23 can be rewritten in the form

$$\begin{aligned} b_n(x)\, D^n y &+ b_{n-1}(x)\, D^{n-1} y + \cdots + b_1(x)\, Dy + b_0(x)\, y \\ &= \left[b_n(x)\, D^n + b_{n-1}(x)\, D^{n-1} + \cdots + b_1(x)\, D + b_0(x)\right] y \\ &= p(D)\, y \end{aligned}$$

Thus, the linear differential equation is of the form

$$p(D)\, y = R(x)$$

where $p(D)$ is a polynomial in D with coefficients $b_i(x)$. For such an equation, the general solution has the form

$$y(x) = y_h(x) + y_p(x)$$

where $y_h(x)$ is the homogeneous solution — that is, $p(D)\, y_h = 0$, and $y_p(x)$ is a particular solution.

A linear system is modeled by linear differential equations. For a linear system, the mathematical form of the **state model** is as follows:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \quad \text{State equations} \\ \mathbf{y}(t) &= C\mathbf{x}(t) + D\mathbf{u}(t) \quad \text{Output equations} \end{aligned}$$

where $\mathbf{x}(t)$ is the $n \times 1$ **state vector**; A is an $n \times n$ matrix and B is an $n \times k$ matrix; $\mathbf{u}(t)$ is the $k \times 1$ **input vector**; C is an $m \times n$ matrix, and D is an $m \times k$ matrix. Thus, for a linear system we have the power of linear algebra and matrix theory at our disposal.

Example 2.1 (Motion of an automobile) A classical example of a simplified control system is the motion of a car subject to acceleration and braking controls. A simplified mathematical model of such a system is

$$\frac{d^2s}{dt^2} + a\frac{ds}{dt} + bs = f(t)$$

where $s(t)$ represents position at time t, so that $\frac{ds(t)}{dt}$ represents velocity and $\frac{d^2s(t)}{dt^2}$ represents acceleration. The basic idea of the state variable approach is to select variables that represent the state of the system. Certainly, the position $s(t)$ and the velocity $\frac{ds(t)}{dt}$ both represent states of the system. If we let $x_1(t) = s(t)$, then we are assigning a state variable $x_1(t)$ to represent the position of the system. The velocity $\dot{x}_1(t) = \dot{s}(t)$ can be assigned another state variable

$$x_2(t) = \dot{x}_1(t)$$

This is one of the state equations. Here, we have expressed one state in terms of the other. Proceeding further, note that $\ddot{x}_1(t) = \dot{x}_2(t) = \ddot{s}(t)$ yields the acceleration term. From the second-order differential equation $\ddot{s}(t) = -a\dot{s}(t) - bs(t) + f(t)$, we have

$$\dot{x}_2(t) = -ax_2(t) - bx_1(t) + f(t)$$

This is the second of the state equations. For an n^{th} order differential equation there must be n first-order state equations. In this case, for a second-order differential equation, we have two first-order differential equations. Casting these two equations in vector-matrix form, we can write the set of state equations as

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -b & -a \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$

that is of the form

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$$

where $\mathbf{u}(t) = f(t)$.

To obtain both the position and the velocity of the system as outputs, we can select $y_1(t)$ and $y_2(t)$ to represent the states $x_1(t)$ and $x_2(t)$, respectively. Placing these quantities in vector-matrix form, we obtain the output equation

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} f(t)$$

that is of the form

$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t)$$

Note again that the outputs are expressed in terms of the system states.

2.3 Controllability and observability

An important first step in solving control problems is to determine whether the desired objective can be achieved by manipulating the chosen control variables. Intuitively, a control system should be designed so that the input can bring it from any state to any other state in a finite time, and also so that all the states can be determined from measuring the output variables. The concepts of *controllability* and *observability* formalize these ideas.

A plant is **controllable** if at any given instant, it is possible to control each state in the plant so that a desired outcome can be reached. In the case where a mathematical model of the system is available in the form

$$\dot{\mathbf{x}}(t) = F(\mathbf{x}, u, t) \tag{2.24}$$

the system $\dot{\mathbf{x}}(t)$ is said to be **completely controllable** if for any t_0, any initial condition $\mathbf{x}_0 = \mathbf{x}(t_0)$, and any final state $\mathbf{x}_f$, there exists a finite time T and a control function $u(t)$ defined on the interval $[t_0, T]$ such that $\mathbf{x}(T) = \mathbf{x}_f$. Note that $\mathbf{x}(T)$ is the solution of Equation 2.24 and clearly depends on the function $u(t)$. It can be shown that a linear, time-invariant system

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{2.25}$$

is completely controllable if and only if the $n \times nm$ **controllability matrix**

$$W = \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \tag{2.26}$$

has rank n, where A is $n \times n$ and B is $n \times m$. More generally, the system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) \\ \mathbf{y}(t) &= C(t)\mathbf{x}(t) \end{aligned} \tag{2.27}$$

with A a continuous $n \times n$ matrix, is completely controllable if and only if the $n \times n$ symmetric **controllability matrix**

$$W(t_0, t_1) = \int_{t_0}^{t_1} X(t) X^{-1}(t_0) B(t) B^T(t) \left(X^{-1}\right)^T (t_0) X^T(t)\, dt \tag{2.28}$$

is nonsingular, where $X(t)$ is the unique $n \times n$ matrix satisfying

$$\frac{dX(t)}{dt} = A(t) X(t), \; X(0) = I \tag{2.29}$$

Other types of controllability can be defined. For example, **output controllability** requires attainment of arbitrary final output. The ability to control the state gives rise to the notion that the output (response) of a system may also be controllable, based on the assumption that if all the individual states in a system are controllable, and the output is a linear combination of the states, then the output must also be controllable. Generally, however, there is no guarantee

that a system that exhibits state controllability will also exhibit output controllability. For example, if the output is a linear combination of two or more states, and the states are not independent, then the system may not exhibit output controllability.

We need to keep in mind that controllability is a black and white issue. A model of a plant is either controllable in a given sense or it is not. Clearly, to know that a plant is uncontrollable is a very useful piece of information. However, to know that something is controllable really tells us nothing about the degree of difficulty in achieving the desired objectives. From a practical point of view, we would, of course, also like to know how to check the controllability of a given system.

A plant is **observable** if states can be determined from output observations. Observability therefore, is concerned with the issue of what can be said about the system state when one is given measurements of the plant output. In the case where the mathematical model of the system is available in the form

$$\dot{\mathbf{x}}(t) = F(\mathbf{x}, u, t)$$

the system is said to be **completely observable** if for any initial state $\mathbf{x}(0)$ there is a finite time $T > 0$ for which $\mathbf{x}(0)$ can be uniquely deduced from the output $\mathbf{y} = G(\mathbf{x}, u, t)$ and the input $u(t)$ over the interval $[0, T]$. Measuring the response of an observable system allows one to formulate appropriate control actions that can steer the system to its desired output. For many systems, some system states cannot be determined by observing the output.

The output $\mathbf{y}(t)$ of a system can be measured. The question is whether we can reconstruct the initial condition from this measured output. It can be shown that a time-invariant system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) \end{aligned}$$

is completely observable if and only if the $nr \times n$ **observability matrix**

$$V = \begin{bmatrix} C & CA & CA^2 & \cdots & CA^{n-1} \end{bmatrix}^T$$

has rank n, where A is $n \times n$ and C is $n \times r$. A system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) \\ \mathbf{y}(t) &= C(t)\mathbf{x}(t) \end{aligned}$$

with $A(t)$ continuous, is completely observable if and only if the symmetric **observability matrix**

$$V(t_0, t_1) = \int_{t_0}^{t_1} X(\tau) X^{-1}(t_0) C^T(\tau) \Phi(\tau, t_0) \, d\tau$$

is nonsingular, where $X(\tau)$ is the unique $n \times n$ matrix satisfying

$$\frac{dX}{dt} = A(t) X(t), \; X(0) = I$$

Once again the property of observability is also a black and white issue. A system either is or is not observable. A system that is observable can provide the necessary conditions of the plant variables as described above. However, the observed plant variables may not be sufficient to reconstruct the entire plant dynamics.

There is a "duality" result that connects these two concepts. Namely, a linear system whose state model is of the form

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t) \\ \mathbf{y}(t) &= C(t)\mathbf{x}(t) \end{aligned}$$

where A, B, and C are matrices of appropriate sizes, is completely controllable if and only if the "dual" system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= -A^T(t)\mathbf{x}(t) + C^T(t)\mathbf{u}(t) \\ \mathbf{y}(t) &= B^T(t)\mathbf{x}(t) \end{aligned}$$

is completely observable. This result is related to the fact that a matrix and its transpose have the same rank.

2.4 Stability

Stability analysis of a system to be controlled is the first task in control design. In a general descriptive way, we can think of stability as the capacity of an object to return to its original position, or to equilibrium, after having been displaced. There are two situations for stability: (1) the plant itself is stable (before the addition of a controller), and (2) the closed-loop control system is stable. All controlled systems must be designed to be stable regardless of the stability or instability of the plant. Controllers must be able to handle disturbances that are not characterized by the model of the system, such as a gust of wind acting on a car set to travel at a constant velocity, wind shear acting on a plane in flight, and so on. This is illustrated in Figure 2.10.

The notion of stability can be viewed as a property of a system that is continuously in motion about some equilibrium point. A point position a is called an **equilibrium point** of Equation 2.30 if $f(\mathbf{a}, t) = 0$ for all t. By changing variables, $\mathbf{y} = \mathbf{x} - \mathbf{a}$, the equilibrium point $\mathbf{a}$ can be transferred to the origin. By this means, you can assume that $\mathbf{a} = \mathbf{0}$. Thus, we will always refer to the stability at the point $\mathbf{0}$. The stability of a dynamical system that is described by a differential equation of the form

$$\dot{\mathbf{x}} = \frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}, t) \tag{2.30}$$

is referred to as **stability about an equilibrium point**.

When $\mathbf{0}$ is an equilibrium state of the system, the system will remain at $\mathbf{0}$ if started from there. In other words, if $\mathbf{x}(t_0) = \mathbf{0}$, then $\mathbf{x}(t) = \mathbf{0}$ for all $t \geq t_0$. This is the intuitive idea of an equilibrium state.

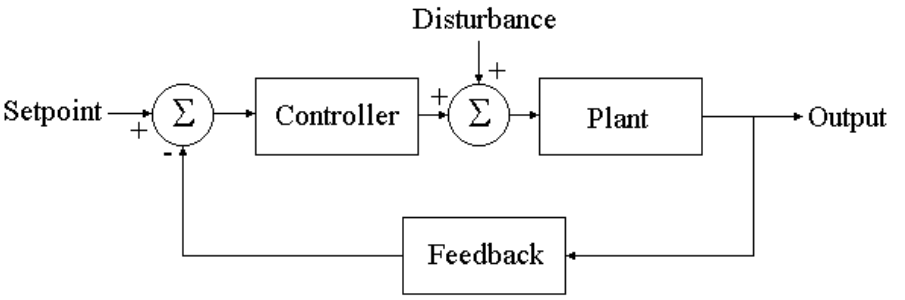


Figure 2.10. Disturbance in system

In general, a dynamical system can have several equilibrium states. Also, the concept of stability about an equilibrium state can be formulated in many different ways. Below is a popular concept of stability.

Definition 2.1 *The equilibrium state* $\mathbf{0}$ *of Equation 2.30 is said to be*

1. ***stable*** *(in the sense of Lyapunov) if for all* $\varepsilon > 0$*, there exists* $\delta > 0$ *such that if* $\|\mathbf{x}(t_0)\| < \delta$ *then* $\|\mathbf{x}(t)\| < \varepsilon$ *for all* $t \geq t_0$*.*

2. ***asymptotically stable*** *if* $\mathbf{0}$ *is stable and* $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{0}$*.*

3. ***asymptotically stable in the large*** *if* $\mathbf{0}$ *is asymptotically stable and* $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{0}$ *regardless of how large are the perturbations around* $\mathbf{0}$*.*

In this definition we use $\|\mathbf{x}(t)\|$ to denote the Euclidean norm, noting that the state space is some $\mathbb{R}^n$. Of course $\mathbf{0}$ is unstable if there is an $\varepsilon > 0$ such that for all $\delta > 0$ there exists $\mathbf{x}(t_0)$ with $\|\mathbf{x}(t_0)\| < \delta$ and $\|\mathbf{x}(t_1)\| > \varepsilon$ for some $t_1 \geq t_0$.

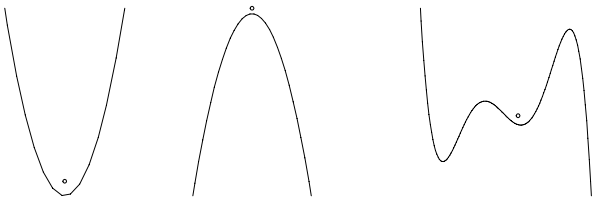


Figure 2.11. Equilibrium points

The notions of *stable* and *asymptotically stable* refer to two different properties of stability of 0. In other words, the nature of stability may vary from one equilibrium point to another. The intuitive idea for stability is clear: for small

perturbations, from the equilibrium 0 at some time t_0, the system remains close to it in subsequent motion. This concept of stability is due to Lyapunov, and often referred to as "stability in the sense of Lyapunov."

In Figure 2.11, the figure on the left represents stability in the sense of Lyapunov if friction is ignored, and asymptotic stability if friction is taken into account, whereas the figure in the center represents instability. The figure on the right represents stability, which is a local condition. In the figure on the left, even if friction is present, a ball would eventually return to equilibrium no matter how large the disturbance. This is an illustration of *asymptotic stability in the large.*

2.4.1 Damping and system response

A control system produces an output, or response, for a given input, or stimulus. In a stable system, the initial response until the system reaches steady state is called the **transient response**. After the transient response, the system approaches its steady-state response, which is its approximation for the commanded or desired response. The nature and duration of the transient response are determined by the *damping* characteristics of the plant.

The possibility exists for a transient response that consists of damped oscillations — that is, a sinusoidal response whose amplitude about the steady-state value diminishes with time. There are responses that are characterized as being **overdamped** (Figure 2.12 (a)) or **critically damped** (Figure 2.12 (b)). An

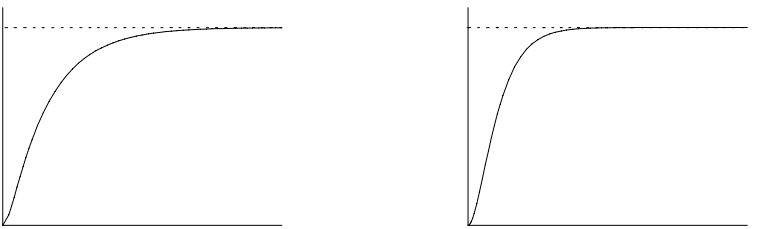


Figure 2.12. (a) Overdamped response (b) Critically damped response

overdamped system is characterized by no overshoot. This occurs when there is a large amount of energy absorption in the system that inhibits the transient response from overshooting and oscillating about the steady-state value in response to the input. A critically damped response is characterized by no overshoot and a rise time that is faster than any possible overdamped response with the same natural frequency. Both are considered stable because there is a steady-state value for each type of response. Stated differently, the system is in "equilibrium." This equilibrium condition is achieved even if the system is allowed to oscillate a bit before achieving steady state. Systems for which

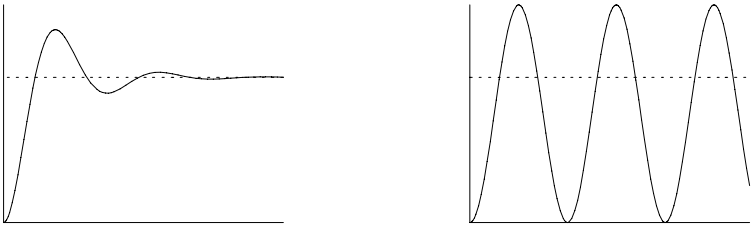


Figure 2.13. (a) Underdamped response (b) Undamped response

the initial response is to oscillate before achieving steady state are referred to as **underdamped systems** (Figure 2.13 (a)). An underdamped response is characterized by overshoot, and an **undamped response** (Figure 2.13 (b)) by sustained oscillation.

A certain amount of oscillation is tolerable in the system response. For example, if a change in the output of a plant is desired, the input to the plant is changed in the form of a step change. Upon receiving this step change in input, we expect the plant to respond quickly so that the desired output can be obtained and maintained as rapidly as possible. We can let the plant output have a fast rate of rise so that the desired output can be achieved quickly. In doing so, we need to allow for a small overshoot and then control the response to exhibit a frequency of oscillation that is adequately damped to bring the plant response towards the desired value in the shortest possible time. A detailed description of the response characteristics of the system is necessary both for analysis and design.

2.4.2 Stability of linear systems

Consider the special case

$$f(\mathbf{x}, t) = A\mathbf{x}(t) \tag{2.31}$$

where A is a constant $n \times n$ matrix. If A is nonsingular, that is, if $\det A \neq 0$, then the system described by Equation 2.31 has a unique equilibrium point, namely 0. For this situation, we can simply talk about the stability of the linear system. Its analysis is based upon the following theorem.

Theorem 2.1 *The linear system $\dot{\mathbf{x}} = A\mathbf{x}$ is asymptotically stable if and only if all eigenvalues of the matrix A have negative real parts.*

Proof. The solution of $\dot{\mathbf{x}} = A\mathbf{x}$ is

$$\mathbf{x}(t) = \mathbf{x}_0 e^{At}$$

where

$$e^{At} = \sum_{k=0}^{\infty} \frac{t^k}{k!} A^k$$

with A^0 the identity $n \times n$ matrix and $\mathbf{x}_0 = \mathbf{x}(0)$. Now

$$e^{At} = \sum_{k=1}^{m} \left[B_{n_1} + B_{n_2} t + \cdots + B_{n_{\alpha_k}} t^{\alpha_k - 1} \right] e^{\lambda_k t} \tag{2.32}$$

where the λ_ks are eigenvalues of A, the α_ks are coefficients of the minimum polynomial of A, and the B_ns are constant matrices determined solely by A.

Thus,

$$\begin{aligned} \left\| e^{At} \right\| &\leq \sum_{k=1}^{m} \sum_{i=1}^{\alpha_k} t^{i-1} \left\| e^{\lambda_k t} \right\| \left\| B_{n_i} \right\| \\ &= \sum_{k=1}^{m} \sum_{i=1}^{\alpha_k} t^{i-1} e^{\mathrm{Re}(\lambda_k) t} \left\| B_{n_i} \right\| \end{aligned}$$

where $\mathrm{Re}(\lambda_k)$ denotes the real part of λ_k.

Thus, if $\mathrm{Re}(\lambda_k) < 0$ for all k, then

$$\lim_{t \to \infty} \|x(t)\| \leq \lim_{t \to \infty} \|x_0\| \left\| e^{At} \right\| = 0$$

so that the origin is asymptotically stable.

Conversely, suppose the origin is asymptotically stable. Then $\mathrm{Re}(\lambda_k) < 0$ for all k, for if there exists some λ_k such that $\mathrm{Re}(\lambda_k) > 0$, then we see from Equation 2.32 that $\lim_{t \to \infty} \|x(t)\| = \infty$ so that the origin is unstable. ■

Such a matrix, or its characteristic polynomial, is said to be **stable**.

Example 2.2 The solution of a second-order constant-coefficient differential equation of the form

$$\frac{d^2 y}{dt^2} + a \frac{dy}{dt} + by = 0$$

is stable if the real parts of the roots

$$\begin{aligned} s_1 &= -\frac{1}{2} \left(a - \sqrt{a^2 - 4b} \right) \\ s_2 &= -\frac{1}{2} \left(a + \sqrt{a^2 - 4b} \right) \end{aligned}$$

of the characteristic polynomial $s^2 + as + b$ lie in the left-half s-plane. In practice, the characteristic polynomial is often found by taking the Laplace transform to get the transfer function

$$\mathcal{L}(y) = \frac{y'(0) + (a + s) y(0)}{s^2 + as + b}$$

The roots of $s^2 + as + b$ are called the poles of the rational function $\mathcal{L}(y)$.

If bounded inputs provide bounded outputs, this is called BIBO stability. If a linear system is asymptotically stable, then the associated controlled system is BIBO stable.

2.4.3 Stability of nonlinear systems

Stability analysis for nonlinear systems is more complicated than for linear systems. There is, nevertheless, an extensive theory for control of nonlinear systems, based on Lyapunov functions. This theory depends on the mathematical models of the systems, and when considering fuzzy and neural control as an alternative to standard control of nonlinear systems, we will need to consider other approaches to stability analysis.

For a nonlinear system of the form

$$\dot{\mathbf{x}} = f(\mathbf{x}),\ f(\mathbf{0}) = 0 \tag{2.33}$$

with $\mathbf{x}(t_0) = \mathbf{x}_0$, it is possible to determine the nature of stability of the origin without solving the equation to obtain $\mathbf{x}(t)$. Sufficient conditions for stability of an equilibrium state are given in terms of the *Lyapunov function*. These conditions generalize the well-known property that an equilibrium point is stable if the energy is a minimum.

Definition 2.2 *A **Lyapunov function** for a system $\dot{\mathbf{x}} = f(\mathbf{x})$ is a function $V : \mathbb{R}^n \to \mathbb{R}$ such that*

1. *V and all its partial derivatives $\frac{\partial V}{\partial x_i}$, $i = 1, 2, \ldots, n$ are continuous.*
2. *$V(\mathbf{0}) = 0$ and $V(\mathbf{x}) > 0$ for all $\mathbf{x} \neq \mathbf{0}$ in some neighborhood $\|\mathbf{x}\| < k$ of $\mathbf{0}$. That is, V is **positive definite**.*
3. *For $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$ satisfying $\dot{\mathbf{x}} = f(\mathbf{x})$ with $f(\mathbf{0}) = 0$,*

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial x_1}\dot{x}_1 + \cdots + \frac{\partial V}{\partial x_n}\dot{x}_n$$

*is such that $\dot{V}(\mathbf{0}) = 0$ and $\dot{V}(\mathbf{x}) \leq 0$ for all $\mathbf{x}$ in some neighborhood of $\mathbf{0}$. In other words, $\dot{V}$ is **negative semidefinite**.*

Theorem 2.2 *For a nonlinear system of the form*

$$\dot{\mathbf{x}} = f(\mathbf{x}),\ f(\mathbf{0}) = 0$$

the origin is stable if there is a Lyapunov function V for the system $\dot{\mathbf{x}} = f(\mathbf{x})$, $f(\mathbf{0}) = 0$.

Proof. Take a number $k > 0$ satisfying both $V(\mathbf{x}) > 0$ and $\dot{V}(\mathbf{x}) \leq 0$ for all $\mathbf{x} \neq \mathbf{0}$ in the neighborhood $\|\mathbf{x}\| < k$ of $\mathbf{0}$. Then there exists a continuous scalar function $\varphi : \mathbb{R} \to \mathbb{R}$ with $\varphi(0) = 0$ that is strictly increasing on the interval $[0, k]$ such that

$$\varphi(\|\mathbf{x}\|) \leq V(\mathbf{x})$$

for all $\mathbf{x}$ in the neighborhood $\|\mathbf{x}\| < k$ of $\mathbf{0}$. Given $\varepsilon > 0$, then since $\varphi(\varepsilon) > 0$, $V(0) = 0$ and $V(x)$ is continuous, and $\mathbf{x}_0 = \mathbf{x}(t_0)$ can be chosen sufficiently close to the origin so that the inequalities

$$\|\mathbf{x}_0\| < \varepsilon,\ V(\mathbf{x}_0) < \varphi(\varepsilon)$$

are simultaneously satisfied. Also, since $\dot{V}(\mathbf{x}) \leq 0$ for all $\mathbf{x}$ in the neighborhood $\|\mathbf{x}\| < k$ of $\mathbf{0}$, $t_0 \leq t_1 \leq k$ implies

$$V(\mathbf{x}(t_1)) \leq V(\mathbf{x}(t_0)) < \varphi(\varepsilon)$$

Thus, for all $\mathbf{x}$ in the neighborhood $\|\mathbf{x}\| < k$ of $\mathbf{0}$, $t_0 \leq t_1 \leq k$ implies

$$\|\mathbf{x}(t_1)\| < \varepsilon$$

since we know that $\varphi(\|\mathbf{x}(t_1)\|) \leq V(\mathbf{x}(t_1)) < \varphi(\varepsilon)$, and $\|\mathbf{x}(t_1)\| \geq \varepsilon$ would imply $\varphi(\|\mathbf{x}(t_1)\|) \geq \varphi(\varepsilon)$ by the property that φ is strictly increasing on $[0, k]$. Taking $\delta = \varepsilon$, we see by Definition 2.1 that the origin is stable. ■

The proof of the following theorem is similar. A function $\dot{V} : \mathbb{R}^n \to \mathbb{R}$ is said to be **negative definite** if $\dot{V}(\mathbf{0}) = 0$ and for all $\mathbf{x} \neq 0$ in some neighborhood $\|\mathbf{x}\| < k$ of $\mathbf{0}$, $\dot{V}(\mathbf{x}) < 0$.

Theorem 2.3 *For a nonlinear system of the form*

$$\dot{x} = f(x),\ f(0) = 0 \tag{2.34}$$

the origin is asymptotically stable if there is a Lyapunov function V for the system, with $\dot{V}$ negative definite.

Here is an example.

Example 2.3 *Consider the nonlinear system*

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$$

where

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},\ \dot{\mathbf{x}} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \end{pmatrix}$$

with

$$\begin{aligned} f_1(\mathbf{x}) &= x_1\left(x_1^2 + x_2^2 - 1\right) - x_2 \\ f_2(\mathbf{x}) &= x_1 + x_2\left(x_1^2 + x_2^2 - 1\right) \end{aligned}$$

The origin $(0, 0)$ is an equilibrium position. The positive definite function

$$V(\mathbf{x}) = x_1^2 + x_2^2$$

has its derivative along any system trajectory

$$\begin{aligned} \dot{V}(x) &= \frac{\partial V}{\partial x_1}\dot{x}_1 + \frac{\partial V}{\partial x_2}\dot{x}_2 \\ &= 2x_1\left[x_1\left(x_1^2 + x_2^2 - 1\right) - x_2\right] + 2x_2\left[x_1 + x_2\left(x_1^2 + x_2^2 - 1\right)\right] \\ &= 2\left(x_1^2 + x_2^2 - 1\right)\left(x_1^2 + x_2^2\right) \end{aligned}$$

When $x_1^2 + x_2^2 < 0$, we have $\dot{V}(x) < 0$, so that $(0, 0)$ is asymptotically stable.

2.4.4 Robust stability

The problem of robust stability in control of linear systems is to ensure system stability in the presence of parameter variations. We know that the origin is asymptotically stable if all the eigenvalues of the matrix A have negative real parts. When A is an $n \times n$ matrix, there are n eigenvalues that are roots of the associated characteristic polynomial $P_n(x) = \sum_{k=0}^{n} a_k x^k$. Thus, when the coefficients a_k are known (given in terms of operating parameters of the plant), it is possible to check stability since there are only finitely many roots to check.

These parameters might change over time due to various factors such as wearing out or aging; and hence, the coefficients a_k should be put in tolerance intervals $[a_k^-, a_k^+]$, allowing each of them to vary within the interval. Thus, we have an infinite family of polynomials P_n, indexed by coefficients in the intervals $[a_k^-, a_k^+]$, $k = 0, 1, 2, ..., n$. In other words, we have an **interval-coefficient polynomial**

$$\sum_{k=0}^{n} [a_k^-, a_k^+] x^k$$

This is a realistic situation where one needs to design controllers to handle stability under this type of "uncertainty" — that is, regardless of how the coefficients a_k are chosen in each $[a_k^-, a_k^+]$. The controller needs to be robust in the sense that it will keep the plant stable when the plant parameters vary within some bounds. For that, we need to be able to check the n roots of members of an infinite family of polynomials. It seems like an impossible task. But if that were so, there would be no way to construct controllers to obtain robust stability.

Mathematically, it looks as though we are facing an "infinite problem." However, like some other problems that appear to be infinite problems, this is a *finite problem.* This discovery, due to Kharitonov in 1978, makes robust control possible for engineers to design. Here is an outline of his result.

Theorem 2.4 (Kharitonov) *Suppose $[a_k^-, a_k^+]$, $k = 0, 1, ..., n$ is a family of intervals. All polynomials of the form $P_n(x) = \sum_{k=0}^{n} a_k x^k$, where $a_k \in [a_k^-, a_k^+]$, are stable if and only if the following four Kharitonov canonical polynomials are stable:*

$$\begin{aligned}
K_1(x) &= a_0^- + a_1^- x + a_2^+ x^2 + a_3^+ x^3 + a_4^- x^4 + a_5^- x^5 + a_6^+ x^6 + \cdots \\
K_2(x) &= a_0^+ + a_1^+ x + a_2^- x^2 + a_3^- x^3 + a_4^+ x^4 + a_5^+ x^5 + a_6^- x^6 + \cdots \\
K_3(x) &= a_0^+ + a_1^- x + a_2^- x^2 + a_3^+ x^3 + a_4^+ x^4 + a_5^- x^5 + a_6^- x^6 + \cdots \\
K_4(x) &= a_0^- + a_1^+ x + a_2^+ x^2 + a_3^- x^3 + a_4^- x^4 + a_5^+ x^5 + a_6^+ x^6 + \cdots
\end{aligned}$$

Note that the pattern for producing these four polynomials is obtained by repetitions of the symbol pattern

$$\begin{bmatrix} - & - & + & + \\ + & + & - & - \\ + & - & - & + \\ - & + & + & - \end{bmatrix}$$

by the superscripts of the lower and upper bounds of the intervals.

Example 2.4 *Take polynomials* $P_2(x) = a_0 + a_1 x + a_2 x^2$ *where* $a_0 \in [0.9, 1.1]$, $a_1 \in [-0.9, 0.1]$, *and* $a_2 \in [0.9, 1.1]$. *The four Kharitonov canonical polynomials are*

$$\begin{aligned} K_1(x) &= 0.9 - 0.9x + 1.1x^2 \\ K_2(x) &= 1.1 + 0.1x + 0.9x^2 \\ K_3(x) &= 1.1 - 0.9x + 0.9x^2 \\ K_4(x) &= 0.9 + 0.1x + 1.1x^2 \end{aligned}$$

Two of these polynomials, $K_1(x)$ *and* $K_3(x)$, *have roots with positive real parts,* 0.409 *and* 0.5, *so the interval-coefficient polynomial with these intervals does not represent a stable system. On the other hand, if* $a_0 \in [1.0, 1.1]$, $a_1 \in [0.9, 1.0]$, *and* $a_2 \in [0.9, 1.0]$, *the four Kharitonov canonical polynomials are*

$$\begin{aligned} K_1(x) &= 1 + 0.99x + x^2 \\ K_2(x) &= 1.1 + x + 0.99x^2 \\ K_3(x) &= 1.1 + 0.99x + 0.99x^2 \\ K_4(x) &= 1 + x + x^2 \end{aligned}$$

all of whose roots have negative real parts, and we know that a system producing these polynomials is stable.

2.5 Controller design

There is a two-fold objective in the design of controllers. First, the overall control loop must be stable. Second, the input to the plant must be such that the desired set-point is achieved in minimum time within some specified criteria. Both of these objectives require full knowledge of the plant.[1]

The fundamental objective in feedback control is to make the output of a plant track the desired input. Generally, this is referred to as **set-point control**. The controller can be placed either in the forward path (in series) with the plant or in the feedback path. Figures 2.14 (a) and (b) illustrate such configurations.

Most of the conventional design methods in control systems rely on the so-called **fixed-configuration** design in that the designer at the outset decides the basic configuration of the overall designed system and the location where the controller is to be positioned relative to the controlled process. The problem then involves the design of the elements of the controller. Because most control efforts involve the modification or compensation of the system performance characteristics, the general design using fixed configuration is also called **compensation**.

[1] We are restricting our discussion of classical methods to only those that have a direct bearing on the fuzzy controllers which will be discussed later. The reader should be aware that classical methods exist for many other types of controller designs that are very powerful and have been used extensively in modern control systems.

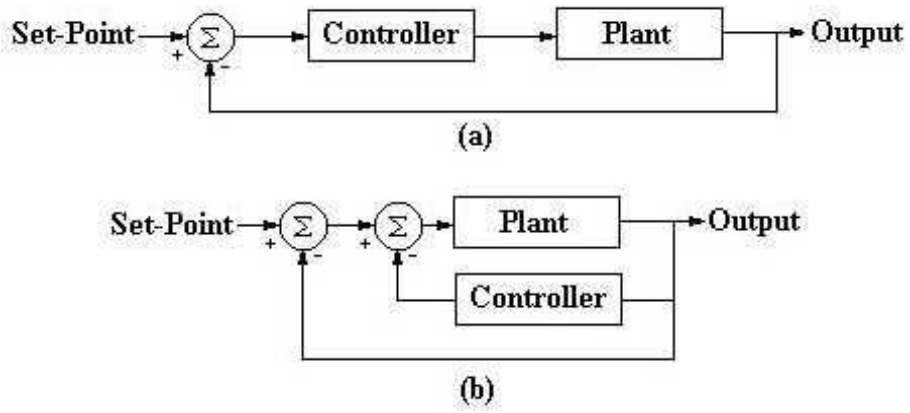


Figure 2.14. Set-point control

The most commonly used configuration is shown in Figure 2.14 (a) in which the controller is placed in series with the plant to be controlled. This type of control is referred to as **series** or **cascade compensation**. The PID controller has a configuration illustrated in Figure 2.14 (a). In Figure 2.14 (b) the compensator is placed in the minor feedback path and is usually referred to as feedback compensation. State-variable and output-feedback controllers have the configuration shown in Figure 2.14 (b).

Standard control is based on knowledge of a mathematical model of the system. The mathematical model can take on basically two forms:

1. The *actual model* of the system.

2. An *estimated model* of the system.

For the classical design of controllers, we need to know something numerical about the system. The so-called numerical approach requires that we know something about the dynamics of the system. As briefly discussed earlier, the model of the plant can be described by a set of equations. Typically, what we have is a set of nonlinear differential equations for which a transfer function does not exist. Designing a controller for a nonlinear system requires that the system first be linearized about some nominal operating point. From this linearized model, a **transfer function** of the plant, that is, the ratio of the output function to the input function, can be derived.

These computations are normally done in the "frequency domain" obtained by taking Laplace transforms of the state equations. The **Laplace transform** of a function $\mathbf{x}(t)$ is defined as

$$\hat{\mathbf{x}}(s) = \int_0^\infty \mathbf{x}(t)\, e^{-st} dt$$

for values of s for which this improper integral converges. Taking Laplace transforms of the system of equations

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A\mathbf{x}(t) + B\mathbf{u}(t) \\ \mathbf{y}(t) &= C\mathbf{x}(t) + E\mathbf{u}(t) \end{aligned} \tag{2.35}$$

with $\mathbf{x}(0) = 0$, we obtain

$$\begin{aligned} s\hat{\mathbf{x}}(s) &= A\hat{\mathbf{x}}(s) + B\hat{\mathbf{u}}(s) \\ \hat{\mathbf{y}}(s) &= C\hat{\mathbf{x}}(s) + E\hat{\mathbf{u}}(s) \end{aligned} \tag{2.36}$$

Writing $s\hat{\mathbf{x}}(s) = sI\hat{\mathbf{x}}(s)$ where I is the identity matrix of the appropriate size, we get

$$(sI - A)\hat{\mathbf{x}}(s) = B\hat{\mathbf{u}}(s) \tag{2.37}$$

and inverting the matrix $sI - A$,

$$\begin{aligned} \hat{\mathbf{y}}(s) &= C(sI - A)^{-1} B\hat{\mathbf{u}}(s) + E\hat{\mathbf{u}}(s) \\ &= \left(C(sI - A)^{-1} B + E\right)\hat{\mathbf{u}}(s) \end{aligned} \tag{2.38}$$

Thus, the **system transfer function**

$$G(s) = C(sI - A)^{-1} B + E \tag{2.39}$$

satisfies

$$G(s)\hat{\mathbf{u}}(s) = \hat{\mathbf{y}}(s) \tag{2.40}$$

and thus describes the ratio between the output $\hat{\mathbf{y}}(s)$ and the input $\hat{\mathbf{u}}(s)$. The ratio $\dfrac{C(s)}{R(s)} = \dfrac{\hat{\mathbf{y}}(s)}{\hat{\mathbf{u}}(s)}$ is also known as the **closed-loop control ratio**.

There are a number of other reasons why transfer functions obtained from the Laplace transform are useful. A system represented by a differential equation is difficult to model as a block diagram, but the Laplace transform of the system has a very convenient representation of this form. Another reason is that in the solution $\mathbf{x}(t) = e^{At}\mathbf{x}(0) + \int_0^t e^{A(t-\tau)} B\mathbf{u}(\tau)\, d\tau$ to the differential equation model in Equation 2.35, the input $\mathbf{u}$ appears inside the "convolution integral," whereas in the transfer model, $\hat{\mathbf{x}}(s)$ becomes a rational function multiplied by the input $\hat{\mathbf{u}}$. Also, the transfer function in the frequency domain is analytic, except at a finite number of poles, and analytic function theory is very useful in analyzing control systems.

In the closed-loop feedback system shown in Figure 2.15, $R(s) = \hat{\mathbf{u}}(s)$ is the **reference input**, $C(s) = \hat{\mathbf{y}}(s)$ is the **system output**, $G(s)$ is the closed-loop control ratio, $E(s)$ is the **error**, and $H(s)$ is the **feedback transfer function**. From this figure we can derive the following relationships:

1. The output of the plant is $C(s) = G(s)E(s)$.

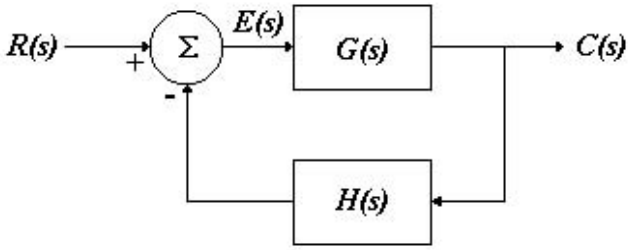


Figure 2.15. Closed-loop feedback system

2. The error between the input value $E(s)$ and the output $C(s)$ of the system amplified or reduced by the feedback transfer function $H(s)$ is

$$E(s) = R(s) - H(s)C(s)$$

3. Substituting the error $E(s)$ into the equation for output of the plant, we get

$$C(s) = G(s)[R(s) - H(s)C(s)]$$

4. Collecting terms and rearranging, we get

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)}$$

that we can write as

$$\frac{G(s)}{1 + G(s)H(s)} = \frac{N(s)}{D(s)}$$

where $N(s)$ and $D(s)$ are polynomials in the variable s.

Factoring $N(s)$ displays all the zeros of the plant transfer function; factoring $D(s)$ displays all the poles of the plant transfer function. The roots of $D(s)$ are called the **characteristic roots** of the system; $D(s) = 0$ is the **characteristic equation**.

A knowledge of the poles and zeros of the plant transfer function provides the means to examine the transient behavior of the system, and to obtain limits on the loop gain to guarantee stability of the system. If a system is inherently unstable, such as the case of an inverted pendulum, the plant transfer function provides the designer with information on how to compensate the pole-zero behavior of the plant with that of a controller. In this case, the controller poles and zeros are selected in combination with the plant poles and zeros to produce a transient response that meets the necessary design requirements.

Knowledge of the plant mathematical model clearly provides the means to obtain the mathematical form of the controller. We can choose to place the

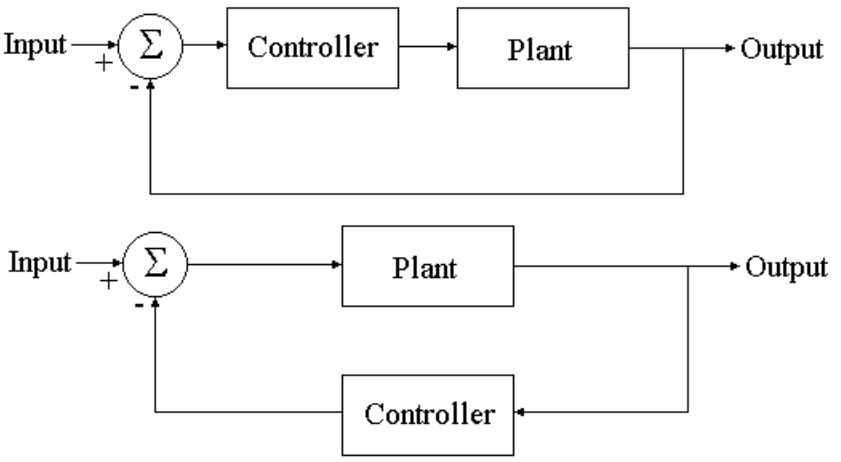


Figure 2.16. Controller in forward or feedback path

controller either in the forward path or in the feedback path as illustrated in Figure 2.16. Once the structure of the control system is decided upon, suitable parameters can then be chosen for the controller model to obtain the desired plant response characteristics.

Consider for example, the closed-loop transfer function of an unstable plant given by the control ratio

$$G_p(s) = \frac{C(s)}{R(s)} = \frac{1}{(s+1)(s-2)}$$

where $C(s)$ is the plant output and $R(s)$ is the plant input. Upon examining the roots, it is clear that the system is indeed unstable — that is, the root satisfying $s - 2 = 0$ lies in the right-half s-plane. Suppose we wish to develop a controller such that the system becomes stable and has specific response characteristics. Let $G_c(s)$ and $G_p(s)$ represent the controller and plant transfer functions, respectively. If the controller is in the forward path, then the closed-loop transfer function would be given as

$$G_1(s) = \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)}$$

Substituting the given plant transfer function and rearranging yields

$$G_1(s) = \frac{G_c(s)}{(s+1)(s-2) + G_c(s)}$$

The denominator of this new control ratio represents the characteristic equation of the system that includes both the controller and the plant. We are now in a

position to choose any desired set of roots that yield satisfactory performance. Suppose we choose the desired characteristic equation as

$$(s+2)(s+3) = 0$$

Equating the denominator of $G_1(s)$ with the desired characteristic equation yields

$$(s+1)(s-2) + G_c(s) = (s+2)(s+3)$$

Expanding and simplifying yields the controller mathematical form

$$G_c(s) = (6s+8)$$

Suppose we choose the feedback path for the controller design. In this case the resulting closed-loop transfer function would be given as

$$G_2(s) = \frac{G_p(s)}{1 + G_p(s)G_c(s)}$$

Substituting the given plant transfer function and rearranging yields

$$G_2(s) = \frac{1}{(s+1)(s-2) + G_c(s)}$$

Once again, if we choose the desired characteristic equation as $(s+2)(s+3) = 0$, we obtain the same result as before, namely,

$$G_c(s) = (6s+8)$$

In either case, the controller is of the form

$$G_c(s) = K_D s + K_P$$

This clearly has meaning in terms of derivative and proportional gain control. More will be discussed along these lines in later sections. The point we wish to make here is that given the mathematical model of a plant, it is entirely possible to obtain a suitable controller design. What we have seen in this example is the addition of terms to the overall closed-loop transfer function such that some desired performance criteria can be met. In the first case where the controller is placed in the forward path, the resulting closed-loop transfer function of the control system is

$$G_1(s) = \frac{6s+8}{s^2+5s+6}$$

where the numerator term $6s + 8$ represents the addition of a "zero" to the transfer function. Such manipulations of the plant mathematical models to yield desired plant responses is characterized in classical control techniques as **root locus analysis** and the criterion used to adjust the open-loop sensitivity gain is called the **Routh-Hurwitz stability criterion**. Adjusting the open-loop sensitivity gain, and placing limits on the range of this gain will assure stable operation of the system within prescribed limits. We urge the reader to refer to one of many excellent control systems books in the literature for a complete treatise on this topic. See [16] for example.

2.6 State-variable feedback control

Here we address a controller design technique known as state-variable feedback control. The concept behind state-variable feedback control is to determine the poles of the transfer function of the closed-loop system and make changes so that new poles are assigned that meet some design criteria. We will see that the basic solution is to place poles in the controller such that the root loci of the overall system is moved into the left-half s-plane. As mentioned earlier, the objective in control systems design is to stabilize an otherwise unstable plant. The method by which an inherently unstable system can be made stable by reassigning its poles to lie in the left-half s-plane is referred to as **pole placement**. The theory behind this technique comes from the result in Theorem 2.1 on page 37 that states that the linear system $\dot{\mathbf{x}} = A\mathbf{x}$ is asymptotically stable if and only if all eigenvalues of the matrix A have negative real parts, and the fact that these eigenvalues are the poles of the transfer function.

It is not necessary that only unstable systems be considered for such control. Even stable systems that do not meet the criteria for a suitable transient response can be candidates for state-variable feedback control. However, it is necessary that all states of the original system be accessible and that the system be completely controllable.

2.6.1 Second-order systems

We can discuss state-variable feedback control with some simple examples. These examples are chosen only to demonstrate how the behavior of a system can be modified and are not intended to provide a complete theoretical evaluation of state-variable feedback control. There are excellent references that provide full details of this type of control, such as [16] and [28], and the reader is strongly urged to refer to the literature on this topic.

Example 2.5 Consider a second-order system represented by the state and output equations as follows:

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} -1 & 3 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \end{aligned}$$

These equations are of the form $\dot{x} = Ax + Bu$, the state equation, and $y = Cx$, the output equation. The eigenvalues of this system can be determined by considering $\det[\lambda I - A] = 0$, the characteristic polynomial of the system. This gives

$$\det\left(\lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} -1 & 3 \\ 0 & 2 \end{bmatrix}\right) = 0$$

that simplifies to

$$\det\left(\begin{bmatrix} \lambda + 1 & -3 \\ 0 & \lambda - 2 \end{bmatrix}\right) = (\lambda + 1)(\lambda - 2) = 0$$

The characteristic polynomial therefore is $(\lambda + 1)(\lambda - 2) = 0$. Clearly this system is unstable because $(\lambda - 2) = 0$ yields a pole in the right-half s-plane. The objective now is to obtain a new system such that the closed-loop system has poles that are only in the left-half s-plane.

Let us now consider the case in which all system states $\mathbf{x}$ of the system are fed back through a feedback matrix K, where r is the input as illustrated in Figure 2.17.

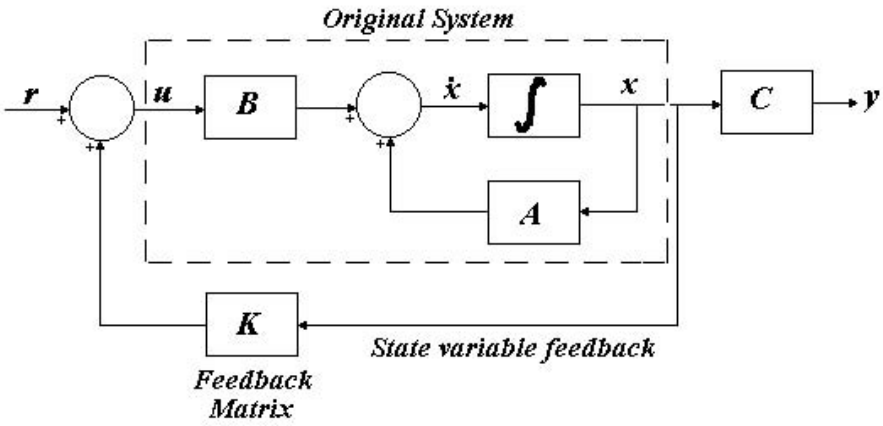


Figure 2.17. State variable feedback control system

For a single-input system with n states, the matrix K is a row vector of dimension $(1 \times n)$, and a control law can be formulated where $u = r + K\mathbf{x}$. Consequently, the state equations can now be written as $\dot{\mathbf{x}} = A\mathbf{x} + B(r + K\mathbf{x})$, which yields $\dot{\mathbf{x}} = (A + BK)\mathbf{x} + Br$. Of course the output remains unchanged. The characteristic polynomial now is $\det(\lambda I - (A + BK)) = 0$. Letting the vector $K = \begin{bmatrix} k_1 & k_2 \end{bmatrix}$ for our example and substituting the matrix elements for A and B, we obtain

$$\det\left(\lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \left(\begin{bmatrix} -1 & 3 \\ 0 & 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} k_1 & k_2 \end{bmatrix}\right)\right) = 0$$

which can be simplified to

$$\det\left(\begin{bmatrix} \lambda + 1 - k_1 & -3 - k_2 \\ -k_1 & \lambda - 2 - k_2 \end{bmatrix}\right) = 0$$

The characteristic polynomial is therefore

$$\lambda^2 + (-1 - k_1 - k_2)\lambda + (-2.0 - k_1 - k_2) = 0$$

Suppose now that the desired characteristic equation is

$$(\lambda + 1)(\lambda + 2) = 0$$

or

$$\lambda^2 + 3\lambda + 2 = 0$$

Comparing the coefficients, we get $(-1 - k_1 - k_2) = 3$ and $(-2.0 - k_1 - k_2) = 2$, which yields two identical equations $(k_1 + k_2) = -4$. While this results in a nonunique solution for the feedback gains k_1 and k_2, we see that any set of parameters satisfying $(k_1 + k_2) = -4$ will shift the pole originally located in the right-half s-plane, namely $(\lambda - 2) = 0$, to the left-half s-plane at $(\lambda + 2) = 0$.

2.6.2 Higher-order systems

Implementing state-variable feedback control for higher-order systems is simplified by first converting the given system to a controller canonical form in which the elements of the characteristic matrix have a prescribed structure. For example, given the characteristic equation of a system as

$$Q(\lambda) = \lambda^n + \alpha_{n-1}\lambda^{n-1} + \cdots + \alpha_1\lambda + \alpha_0$$

the controller canonical form of the characteristic matrix is

$$A_c = \begin{bmatrix} 0 & 1 & & & & 0 \\ 0 & 0 & 1 & & & 0 \\ & & 0 & & & 0 \\ & & & \ddots & 1 & 0 \\ & & & & 0 & 1 \\ -\alpha_0 & -\alpha_1 & & \cdots & -\alpha_{n-2} & -\alpha_{n-1} \end{bmatrix}$$

The entries in the last row of the A_c matrix are the negatives of the coefficients of the characteristic polynomial. In addition, the input conditioning matrix B_c for the controller canonical realization is of the form

$$B_c = [0 \quad 0 \quad \cdots \quad 0 \quad 1]^T$$

Now, if we choose the feedback matrix

$$K = [k_1 \quad k_2 \quad \cdots \quad k_{n-1} \quad k_n]$$

the resulting matrix obtained from $[A + BK]$ can be written as:

$$A_{cf} = \begin{bmatrix} 0 & 1 & & & & \\ & 0 & 1 & & & \\ & & 0 & & & \\ & & & \ddots & 1 & \\ & & & & 0 & 1 \\ -\alpha_0 + k_1 & -\alpha_1 + k_2 & & \cdots & -\alpha_{n-2} + k_{n-1} & -\alpha_{n-1} + k_n \end{bmatrix}$$

Comparing the coefficients of the given system with the desired system yields the feedback gains. Therefore, the canonical form of system representation is more convenient for determining the feedback gains when dealing with higher-order systems.

Example 2.6 Consider an unstable third-order system given by the set of state and output equations as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 1 & 6 & -3 \\ -1 & -1 & 1 \\ -2 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

We notice that the system is not in canonical form and proceed to develop the controller canonical realization. To do this, we first obtain the transfer function of the given system as

$$\frac{Y(s)}{U(s)} = C\left[sI - A\right]^{-1} B + D$$

This can be done very easily using MATLAB by specifying the A, B, C, and D matrices and using the MATLAB function "ss2tf" to obtain the transfer function of the system. The following MATLAB code illustrates this process:

```
a=[1 6 -3;-1 -1 1;-2 2 0]; %Specify the A matrix
b=[1;1;1]; %Specify the B matrix
c=[0 0 1]; %Specify the C matrix
d=0; %Specify the D matrix
[num,den]=ss2tf(a,b,c,d,1); %Note: the 1 indicates a single-input system
```

The resulting transfer function is

$$\frac{Y(s)}{U(s)} = \frac{s^2 - 13}{s^3 - 3s + 2}$$

The denominator of the transfer function factors as $(s+2)(s-1)^2$, displaying the roots of the characteristic equation and clearly indicating an unstable system. From the method discussed above, the characteristic matrix for the controller canonical realization may be obtained directly as

$$A_c = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & 3 & 0 \end{bmatrix}$$

Suppose the desired set of eigenvalues for the system is

$$\{\lambda_1, \lambda_2 = -1 \pm j, \quad \lambda_3 = -4\}$$

which gives the characteristic polynomial

$$Q(\lambda) = \lambda^3 + 6\lambda^2 + 10\lambda + 8$$

From this, we can determine the desired characteristic matrix as

$$A_{cf} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -8 & -10 & -6 \end{bmatrix}$$

Comparing the last row of coefficients between A_c and A_{cf}, we obtain the feedback gain matrix

$$K = [-6 \quad -13 \quad -6]$$

For the examples considered here, we see a fairly straightforward approach to implementing state-variable feedback control. However, as mentioned previously, all states of the original system must be accessible, and the system must be completely controllable. Therefore, we need to know *a priori* that the system is fully controllable before attempting to implement state-variable feedback control.

Such *a priori* knowledge can be obtained by investigating the rank of the $n \times nm$ controllability matrix

$$\begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$$

formed by the A and B matrices of the original system (see page 32). Recall that an n^{th}-order single-input system is fully controllable if and only if

$$\text{Rank} \begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} = n$$

This says that if the rank of the controllability matrix is less than n, the system is not fully controllable. A system being not fully controllable implies that one or more states of the system are not directly controllable. This is due to pole-zero cancellations in the system transfer function as the next example indicates.

Example 2.7 Consider the state and output equations of a second-order system given by

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Computing the transfer function $G(s) = C\,[sI - A]^{-1}\,B + D$ yields

$$G(s) = \frac{(s+1)}{(s+1)(s+2)}$$

in which the pole and zero located at $s = -1$ cancel, making the system uncontrollable at the eigenvalue $\lambda = -1$. Whether or not the system is fully controllable can also be verified by computing the rank of the controllability matrix as follows:

$$\text{Rank} \begin{bmatrix} 1 & \begin{bmatrix} -2 & 0 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{bmatrix} = \text{Rank} \begin{bmatrix} 1 & -2 \\ 1 & -2 \end{bmatrix} = 1$$

This computation showing Rank < 2 implies the system is not fully controllable, but this does not, by itself, indicate which of the two eigenvalues is uncontrollable.

2.7 Proportional-integral-derivative control

Most industrial systems are controlled by classical proportional-integral-derivative (PID) controllers (including P, PD, and PI). This is done despite the system being nonlinear and despite the fact that the simplicity of the concept often limits the performance. The reason why PID controllers have gained such popularity is that detailed knowledge about the system is not required, but the controller can be tuned by means of simple rules of thumb or by PID "autotuners."

In this section, we demonstrate with several examples the design of controllers using the standard approach. These same examples will be developed using a fuzzy, neural, or neural-fuzzy approach in later chapters. The automobile cruise control and temperature control problems are examples of *regulation* problems, where the fundamental desired behavior is to keep the output of the system at a constant level, regardless of disturbances acting on the system. The servomotor dynamics control problem is a component of a *servo* problem, where the fundamental desired behavior is to make the output follow a reference trajectory.

2.7.1 Example: automobile cruise control system

In this example, we develop a simple model of an automobile cruise control system. The control objective is to maintain a speed preset by the driver. If we

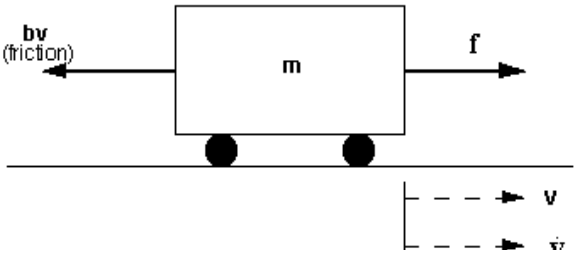


Figure 2.18. Mass and damper system

neglect the inertia of the wheels, and assume that friction (which is proportional to the car's speed) is what is opposing the motion of the car, then the plant description is reduced to the simple mass and damper system shown in Figure

2.18. Using Newton's law of motion, the model equations for this system are

$$\begin{aligned} m\frac{dv}{dt} + bv &= f \\ y &= v \end{aligned}$$

where f is the force produced by the engine and v is the measured velocity of the automobile. For this example, Let us assume that $m = 1000$ kg, $b = 50$ Newton seconds/meter, and $f = 500$ Newtons, a constant force produced by the engine. When the engine produces a 500 Newton force, the car will reach a maximum velocity of 10 meters/second. An automobile should be able to accelerate up to that speed in less than 5 seconds. For this simple cruise control system, we can assume an allowable overshoot of 10% on the velocity and a 2% steady-state error.

The next step is to find the transfer function of the system above. For this, we need to take the Laplace transform of the model equations. When finding the transfer function, zero initial conditions must be assumed. Performing Laplace transforms of the two equations gives

$$\begin{aligned} msV(s) + bV(s) &= F(s) \\ Y(s) &= V(s) \end{aligned}$$

Since our output is the velocity, Let us express $V(s)$ in terms of $Y(s)$ and obtain

$$msY(s) + bY(s) = F(s)$$

The plant transfer function therefore becomes

$$\frac{Y(s)}{F(s)} = \frac{1}{ms + b}$$

Substituting $m = 1000\,\text{kg}$, $b = 50$ Newton seconds/meter assumed for the automobile, the transfer function is

$$\frac{Y(s)}{F(s)} = \frac{1}{1000s + 50} = \frac{(1/1000)}{s + 0.05}$$

For a constant force of 500 Newtons produced from the start, with the automobile initially at rest, this represents a step input where

$$F(s) = \frac{500}{s}$$

We can now compute the open-loop response of the system to examine how well the system behaves without any controller action.

The open-loop response of the plant is that produced by the engine force acting on the mass of the automobile. In this analysis, there is no controller action as there is no feedback. Therefore, the output response is

$$Y(s) = \frac{(1/1000)}{s + 0.05}F(s) = \left[\frac{(1/1000)}{s + 0.05}\right]\left[\frac{500}{s}\right] = \frac{0.5}{s\,(s + 0.05)}$$

Performing partial fractions expansion, we obtain

$$Y(s) = \frac{10}{s} - \frac{10}{s + 0.05}$$

The inverse Laplace transform yields

$$y(t) = 10u(t) - 10e^{-0.05t}u(t)$$

where $u(t)$ is a unit step. The graph in Figure 2.19 shows that the vehicle takes more than 100 seconds to reach the steady-state speed of 10 meters/second. Clearly, this does not satisfy our rise time criterion of less than 5 seconds.

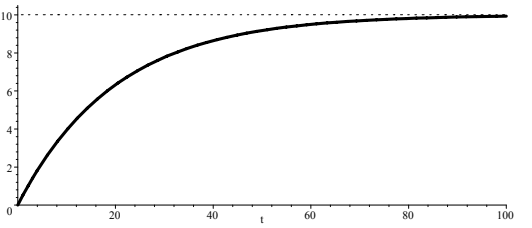


Figure 2.19. $y(t) = 10u(t) - 10e^{-0.05t}u(t)$

From the above analysis, we have determined that a controller is needed to improve the performance. The performance of this system can be improved by providing a unity feedback controller. Figure 2.20 is the block diagram of a typical unity feedback system.

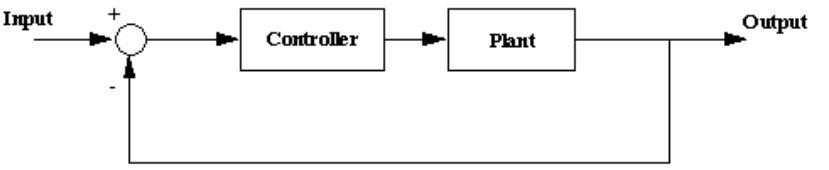


Figure 2.20. Unity feedback system

We choose the standard PID transfer function for the controller, namely,

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s}$$

The plant transfer function is as derived above, namely,

$$G_p(s) = \frac{1}{ms + b}$$

The objective in the controller design then is to select the appropriate parameters for K_P, K_I, and K_D to satisfy all design criteria.

To design a PID controller for a given system, follow the steps shown below to obtain a desired response.

1. Obtain the open-loop response and determine what needs to be improved.
2. Add proportional control to improve the rise time $K_P > 0$.
3. Add derivative control to improve the overshoot $K_D > 0$.
4. Add integral control to eliminate the steady-state error $K_I > 0$.
5. Adjust each of K_P, K_I, and K_D until a desired overall response is obtained.

Working through each of the steps listed above, we have already seen that the open-loop response is not satisfactory in that the rise time is inadequate for the automobile to reach a velocity of 10 meters/second in less than 5 seconds. We must therefore provide some proportional gain, $K_P > 0$, such that the rise time is smaller — that is, the velocity reaches 10 meters/second in less than 5 seconds. In general, it is intuitive to think that if the rise time is made too small, the velocity might reach the desired value at the expense of creating a large overshoot before settling down to the desired value. Naturally then, we need to be concerned about how we can best control the overshoot as a consequence of reducing the rise time.

To achieve only proportional control action, we select $K_P > 0$ and set $K_D = K_I = 0$ in the controller transfer function $G_c(s)$. Selecting values for any of the constants is on a purely trial and error basis. However, we should keep in mind that the proportional gain K_P affects the time constant and consequently the rate of rise.

From the control system diagram, we see that the controller and plant can be combined into a single transfer function, namely,

$$G(s) = G_c(s)G_p(s) = K_P\left(\frac{1}{ms+b}\right)$$

The closed-loop transfer function therefore is determined as

$$\frac{Y(s)}{F(s)} = \frac{G(s)}{1+G(s)H(s)}$$

where $H(s)$ is the feedback transfer function. In this case, since we have chosen a unity feedback system, we set $H(s) = 1$. We therefore obtain the closed-loop transfer function of the system as

$$\frac{Y(s)}{F(s)} = \frac{G(s)}{1+G(s)} = \frac{\frac{K_P}{ms+b}}{1+\frac{K_P}{ms+b}} = \frac{K_P}{ms+b+K_P}$$

Let us select $K_P = 100$ as a start and see what happens. Substituting values for K_P, m, and b we get the closed-loop transfer function as

$$\frac{Y(s)}{F(s)} = \frac{100}{1000s + 50 + 100} = \frac{100}{1000s + 150} = \frac{0.1}{s + 0.15}$$

Since we are interested in obtaining a desired velocity of 10 meters/second, we need to provide a step input of amplitude 10 meters/second. Therefore, using $F(s) = 10/s$, we obtain the closed-loop response $Y(s)$ as,

$$\begin{aligned} Y(s) &= \frac{0.1}{s + 0.15} F(s) = \left(\frac{0.1}{s + 0.15}\right)\left(\frac{10}{s}\right) \\ &= \frac{1}{s\,(s + 0.15)} = \frac{\frac{1}{0.15}}{s} - \frac{\frac{1}{0.15}}{s + 0.15} \\ &= \frac{6.6667}{s} - \frac{6.6667}{s + 0.15} \end{aligned}$$

The time-domain solution for this is $y(t) = 6.6667u(t) - 6.6667e^{-0.15t}u(t)$, shown in Figure 2.21.

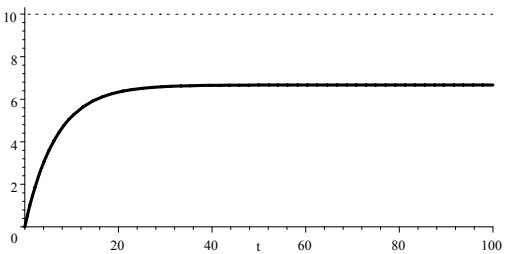


Figure 2.21. Time-domain solution

Once again, we see that choosing $K_P = 100$ still does not satisfy our rise time criterion of less than 5 seconds. However, we do see an improvement over the open-loop response shown earlier. Choosing $K_P = 1000$, and carrying out an analysis similar to the above shows that the closed-loop response satisfies the performance criteria for rise time.

$$\begin{aligned} \frac{Y(s)}{F(s)} &= \frac{1000}{1000s + [50 + 1000]} = \frac{1000}{1000s + 1050} = \frac{1}{s + 1.05} \\ Y(s) &= \frac{1}{s + 1.05} F(s) = \left[\frac{1}{s + 1.05}\right]\left[\frac{10}{s}\right] \\ &= \frac{10}{s\,(s + 1.05)} = \frac{9.9502}{s} - \frac{9.9502}{s + 1.005} \\ y(t) &= 9.9502u(t) - 9.9502e^{-1.005t}u(t) \end{aligned}$$

However, the response continues to exhibit the small steady-state error of $10 - 9.9502 = 0.049\,8$ meters/second shown in Figure 2.22.

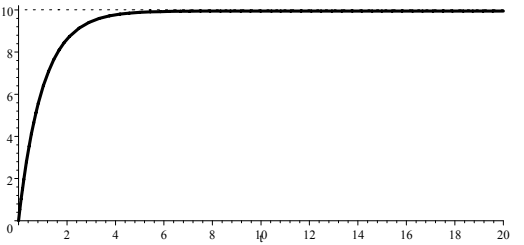


Figure 2.22. $y(t) = 9.9502u(t) - 9.9502e^{-1.005t}u(t)$

We see that there is no issue in this example relating to overshoot. The only issue is that of steady-state error. It should be clear that just increasing the proportional gain will not correct the steady-state error. To correct this, we need to implement integral control. The reason for not needing derivative control should be obvious from the analytical solution for the closed-loop solution; for example, $y(t) = 9.9502u(t) - 9.9502e^{-1.005t}u(t)$ which is of the form $C\left[1 - e^{-at}\right]u(t)$ in which there is no term that can produce sinusoidal oscillation. The shape of the response clearly shows only exponential behavior. Overshoot can occur only if the system oscillates, that is, has second-order complex conjugate poles.

We now proceed to implement integral control by choosing values for $K_I > 0$. Choosing a value of $K_P = 1000$, $K_I = 1$, and with $K_D = 0$, we get the forward transfer function as

$$\begin{aligned} G(s) &= G_c(s)G_p(s) = \left(\frac{K_D s^2 + K_P s + K_I}{s}\right)\left(\frac{1}{ms + b}\right) \\ &= \left(\frac{1000s + 1}{s}\right)\left(\frac{1}{1000s + 50}\right) = \frac{s + 0.001}{s\,(s + 0.05)} \end{aligned}$$

The closed-loop transfer function therefore is given by

$$\frac{Y(s)}{F(s)} = \frac{G(s)}{1 + G(s)} = \frac{\frac{s+0.001}{s(s+0.05)}}{1 + \frac{s+0.001}{s(s+0.05)}} = \frac{s + 0.001}{s^2 + 1.05s + 0.001}$$

A controller canonical form of state-variable realization can then be obtained as follows. Let

$$\frac{Q(s)}{Q(s)}\frac{Y(s)}{F(s)} = \frac{s + 0.001}{s^2 + 1.05s + 0.001}$$

Upon choosing

$$\frac{Q(s)}{F(s)} = \frac{1}{s^2 + 1.05s + 0.001}$$

we obtain a relationship between system state and input $F(s)$, and choosing

$$\frac{Y(s)}{Q(s)} = s + 0.001$$

we obtain a relationship between system state and output $Y(s)$. Here, we are assigning an arbitrary state $Q(s)$. From these two relationships, a state-variable model can be developed.

From the state and input relationship, we obtain

$$(s^2 + 1.05s + 0.001)Q(s) = F(s)$$

The time-domain differential equation, with zero initial conditions is then

$$q''(t) = -1.05q'(t) - 0.001q(t) + f(t)$$

Similarly, from the state and output relationship, we have

$$Y(s) = (s + 0.001)Q(s)$$

The time-domain equation is then

$$y(t) = q'(t) + 0.001q(t)$$

If we choose $x_1(t) = q(t)$, then

$$x_1'(t) = q'(t) = x_2(t)$$

and

$$x_2'(t) = q''(t) = -1.05x_2(t) - 0.001x_1(t) + f(t)$$

represents the set of state-variable equations. The corresponding output equation can then be written as $y(t) = x_2(t) + 0.001x_1(t)$. These equations can be written in standard vector-matrix form as

$$\begin{bmatrix} x_1'(t) \\ x_2'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.001 & -1.05 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$

$$[y(t)] = \begin{bmatrix} 0.001 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

A solution of the set of state and output equations given above can be obtained using a MATLAB simulation package called Simulink. In the simulation diagram in Figure 2.23, we have set up a simulation to obtain the step response of the modified system described earlier. It can be seen in Figure 2.24 that while the rise time is well within the required specification, the steady-state error is large and has to be minimized using integral control action.

Choosing values $K_P = 800$, $K_I = 40$, and $K_D = 0$, we get the forward transfer function as

$$G(s) = G_c(s)G_p(s) = \left(\frac{K_D s^2 + K_P s + K_I}{s}\right)\left(\frac{1}{ms + b}\right)$$

$$= \left(\frac{800s + 40}{s}\right)\left(\frac{1}{1000s + 50}\right) = \frac{0.8}{s}$$

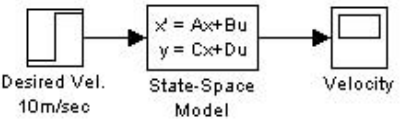


Figure 2.23. Simulation diagram

The closed-loop transfer function therefore is given by

$$\frac{Y(s)}{F(s)} = \frac{G(s)}{1+G(s)} = \frac{\frac{0.8}{s}}{1+\frac{0.8}{s}} = \frac{0.8}{s+0.8}$$

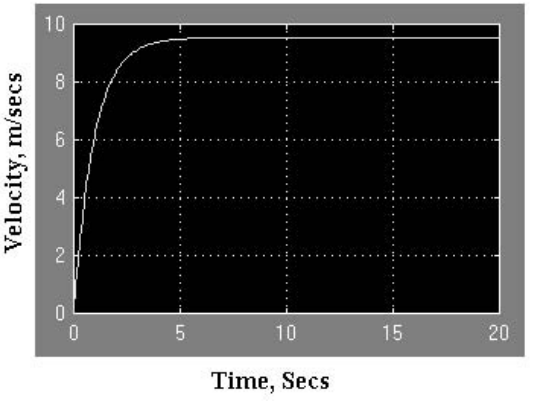


Figure 2.24. Large steady-state error

From this, we compute

$$Y(s) = \frac{0.8}{s+0.8}F(s) = \left(\frac{0.8}{s+0.8}\right)\frac{10}{s} = \frac{10}{s} - \frac{10}{s+0.8}$$

The time-domain solution is

$$y(t) = 10u(t) - 10e^{-0.8t}u(t)$$

as shown in Figure 2.25. From the above example, we have seen the trial and error nature of designing a PID controller. It should become clear that if the model parameters change, so will the parameters of the PID controller. As such, there is significant redesign effort needed to obtain satisfactory performance from the controller.

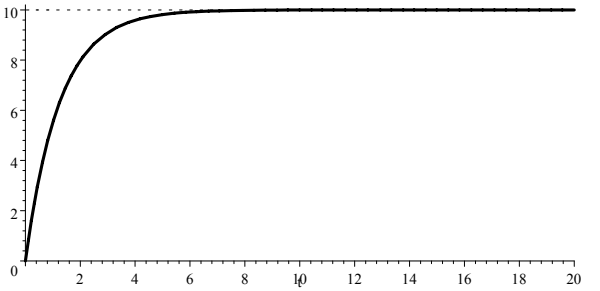


Figure 2.25. $y(t) = 10u(t) - 10e^{-0.8t}u(t)$

2.7.2 Example: temperature control

In this section we discuss the development of the classical proportional-integral-derivative (PID) control parameters for a temperature control problem. We will extend this example in Chapter 6 to incorporate a neural controller. In a conventional PID control system, the gains are all fixed, while with neural networks, they can change.

System model The system comprises an electrical heater of heat capacity C_h connected via a thermal resistance R_{ho} to the oven. The heat capacity of the oven is C_0. At temperature T_e, the oven loses heat to the environment through the thermal resistance R_0 of its insulation. The temperature controller adjusts the power dissipated in the heating elements W, by comparing the oven temperature T_0 with the set-point temperature T_s.

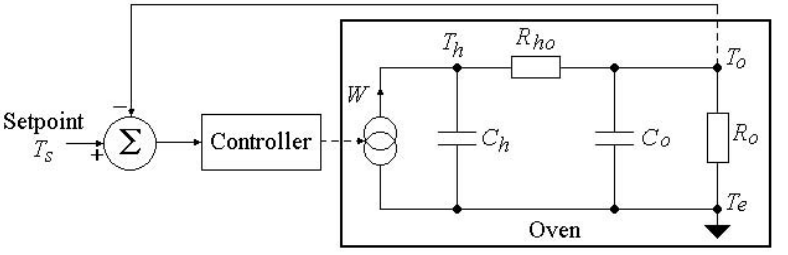


Figure 2.26. System model

The symbols on the right side of the diagram in Figure 2.26 are thermal components; the ones on the left are electrical devices. Dashed lines represent transducers: a thermometer in one case, conversion of electrical current flowing through the heater into heat (thermal current W) in the other. The thermome-

ter time constant is usually very small, so its effects will be assumed to be negligible during much of the following discussion.

We discussed earlier that conventional control requires the use of a system model for the development of a controller. In this context, we develop a state-variable model as follows

$$\begin{bmatrix} \dot{T}_h \\ \dot{T}_o \end{bmatrix} = \begin{bmatrix} -\frac{1}{R_{ho}C_h} & \frac{1}{R_{ho}C_h} \\ \frac{1}{R_{ho}C_o} & -\left(\frac{1}{R_{ho}C_o} + \frac{1}{R_oC_o}\right) \end{bmatrix} \begin{bmatrix} T_h \\ T_o \end{bmatrix} + \begin{bmatrix} \frac{1}{C_h} \\ 0 \end{bmatrix} W$$

$$[T_o] = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} T_h \\ T_o \end{bmatrix}$$

that represents the mathematical model of the system. The heat loss to the environment T_e is a component that cannot be modeled and can only be compensated by supplying sufficient heat by the controlling elements. The set of differential equations (state equations) are driven by the input W that the controller provides. T_o is the oven temperature which is a sensed parameter. The error — that is, the difference between the desired temperature and the sensed temperature, acts as input to the controller. Such a model can very easily be modeled in MATLAB as demonstrated in the following simulations of the system behavior.

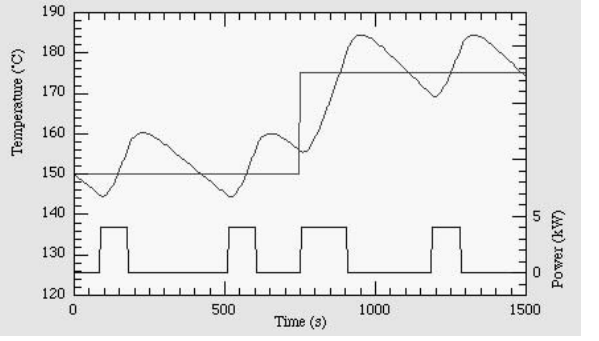


Figure 2.27. Fluctuations in temperature

It is important to mention at the outset that the simplest form of control that can be adopted is the ON-OFF control used by almost all domestic thermostats. When the oven is cooler than the set-point temperature, the heater is turned on at maximum power, and once the oven is hotter than the set-point temperature, the heater is switched off completely. The turn-on and turn-off temperatures are deliberately made to differ by a small amount, known as *hysteresis*, to prevent noise from rapidly and unnecessarily switching the heater when the temperature is near the set-point. The fluctuations in temperature shown in the graph in Figure 2.27 are significantly larger than the hysteresis, due to the significant heat capacity of the heating element. Naturally, the ON-OFF control is inefficient

and results in a high average power consumption. It is for this reason that we need to consider a controller that provides the best control actions that result in the least power consumption.

Proportional control A proportional controller attempts to perform better than the ON-OFF type by applying power W, to the heater in proportion to the difference in temperature between the oven and the set-point

$$W = P(T_s - T_o)$$

where P is known as the "proportional gain" of the controller. As its gain is increased, the system responds faster to changes in set-point, but becomes progressively underdamped and eventually unstable. As shown in Figure 2.28, the final oven temperature lies below the set-point for this system because some difference is required to keep the heater supplying power. The heater power must always lie between zero and the maximum because it can only act as a heat source, and not act as a heat sink.

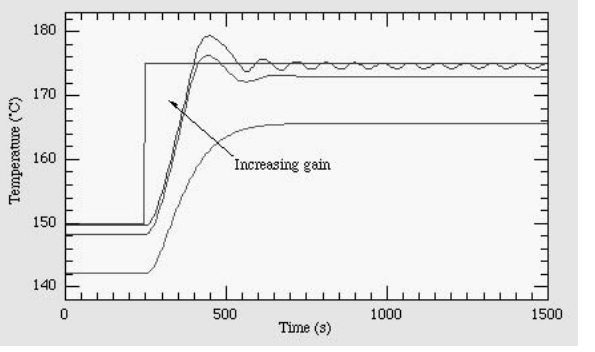


Figure 2.28. Response to proportional control

Proportional + derivative control The stability and overshoot problems that arise when a proportional controller is used at high gain can be mitigated by adding a term proportional to the time-derivative of the error signal,

$$W = P(T_s - T_o) + D\frac{d}{dt}(T_s - T_o)$$

This technique is known as PD control. The value of the damping constant, D, can be adjusted to achieve a critically damped response to changes in the set-point temperature, as shown in Figure 2.29.

It is easy to recognize that too little damping results in overshoot and ringing (oscillation in the oven temperature characterized as underdamped); too much damping causes an unnecessarily slow response characterized as overdamped. As such, critical damping gives the fastest response with little or no oscillation in the oven temperature.

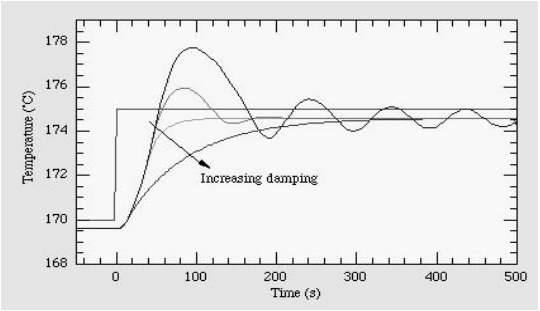


Figure 2.29. Response to PD control

Proportional + integral + derivative control Although PD control deals well with the overshoot and ringing problems associated with proportional control, it does not solve the problem with the steady-state error. Fortunately, it is

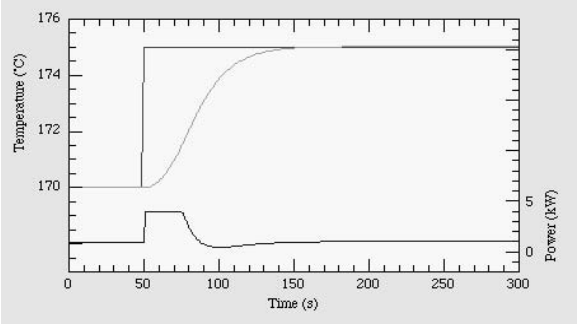


Figure 2.30. Response to PID control

possible to eliminate this while using relatively low gain by adding an integral term to the control function which becomes

$$W = P(T_s - T_o) + D\frac{d}{dt}(T_s - T_o) + I\int (T_s - T_o)dt$$

where I, the *integral gain parameter* is sometimes known as the *controller reset level*. This form of function is known as PID control. The effect of the integral term is to change the heater power until the time-averaged value of the temperature error is zero. The method works quite well but complicates the mathematical analysis slightly because the system is now a third-order system.

Figure 2.30 shows that, as expected, adding the integral term has eliminated the steady-state error. The slight undershoot in the power suggests that there may be scope for further tweaking the PID parameters.

Proportional + integral control Sometimes derivative action can cause the heater power to fluctuate wildly. This can happen when the sensor measuring the oven temperature is susceptible to noise or other electrical interference. In these circumstances, it is often sensible to use a PI controller or set the derivative action of a PID controller to zero.

Third-order systems Systems controlled using an integral action controller are almost always at least third-order. Unlike second-order systems, third-order systems are fairly uncommon in physics, but the methods of control theory make the analysis quite straightforward. For instance, there is a systematic way of classifying the complex roots of the auxiliary equation for the model, known as the Routh-Hurwitz stability criterion. Provided the integral gain is kept sufficiently small, parameter values can be found to give an acceptably damped response, with the error temperature eventually tending to zero, if the set-point is changed by a step or linear ramp in time. Whereas derivative control improved the system damping, integral control eliminates steady-state error at the expense of stability margin.

Using MATLAB for PID controller design The transfer function of a PID controller looks like the following

$$K_p + \frac{K_i}{s} + sK_d = \frac{K_d s^2 + K_p s + K_i}{s}$$

where K_p is the proportional gain, K_i is the integral gain, and K_d is the derivative gain. First, Let us take a look at the effect of a PID controller on the closed-loop system using the schematic in Figure 2.31. To begin, the variable

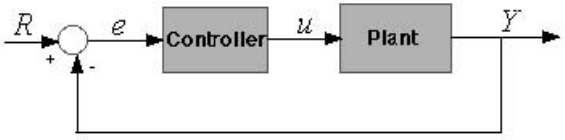


Figure 2.31. Overview of controller and plant

e is the tracking error or the difference between the desired reference value R and the actual output Y. The controller takes this error signal and computes both its derivative and its integral. The signal u that is sent to the actuator is now equal to the proportional gain K_p times the magnitude of the error, plus the integral gain K_i times the integral of the error, plus the derivative gain K_d times the derivative of the error.

Generally speaking, for an open-loop transfer function that has the canonical second-order form

$$\frac{1}{s^2 + 2\zeta\omega_R s + \omega_R^2}$$

a large K_p will have the effect of reducing the rise time and will reduce (but never eliminate) the steady-state error. Integral control K_i, will have the effect of eliminating the steady-state error, but it will make the transient response worse. If integral control is to be used, a small K_i should always be tried first. Derivative control will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response. The effects on the closed-loop response of adding to the controller terms K_p, K_i, and K_d are listed in Table 2.1.

Table 2.1. Effects on closed-loop response

Controller Term	Rise Time	Overshoot	Settling Time	Steady-state Error
K_p	Decreases	Increases	No change	Decreases
K_i	Decreases	Increases	Increases	Eliminates
K_d	No change	Decreases	Decreases	No change

It is important to note that these correlations are to be used only as a guide and not to imply the exact relations between the variables and their effect on each other. Changing one of these variables can change the effect of the other two. The table serves as a guide to allow the designer to adjust the parameters by trial and error.

Simulating the open-loop step response Many PID controllers are designed by the trial and error selection of the variables K_p, K_i, and K_d. There are some rules of thumb that you can consult to determine good values to start from; see your controls book for some explanations of these recommendations.

Suppose we have a second-order plant transfer function

$$G(s) = \frac{1}{s^2 + 10s + 20}$$

Let us first view the open-loop step response. To model this system in MATLAB, create a new *m-file* and add in the following code:

```
numsys=1;
densys=[1 10 20];
step(numsys,densys);
```

A plot of the response is shown in Figure 2.32. From the given transfer function, it is easy to see that the final value of $G(s)$, applying the Final Value theorem, is 1/20, as is illustrated in the step response. This value of 0.05, in relation to the unit step input amounts to a steady-state error of 0.95. Furthermore, the rise time is about 1 second and the settling time is nearly 2 seconds. While it is not clear as to what the control criteria should be, it is evident that at least the steady-state error must be reduced to within some tolerable limits. We proceed now to design a PID controller in a step-by-step manner that illustrates the changes in response that occur as a result of various parameter adjustments.

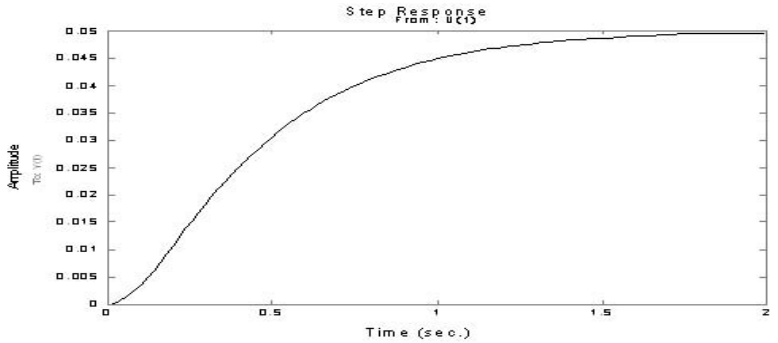


Figure 2.32. Step response of the open-loop transfer function

As outlined previously, the first step is to improve the rise time such that the plant response is at or near the desired level. So, we proceed to first implement proportional control. In the following MATLAB code, we start by inputting a low value of K_d as a start to examine the effect of proportional gain rise time and subsequently on the error. The following MATLAB code implements proportional control:

```
kp=10; %Proportional gain
sysopenloop=tf(kp*numsys,densys); %Open-loop transfer function
sysfeedback=[1]; %Unity feedback
sysclosedloop=feedback(sysopenloop,sysfeedback);%Closed-loop TF
step(sysclosedloop,0:0.001:2.0); %Step input response
```

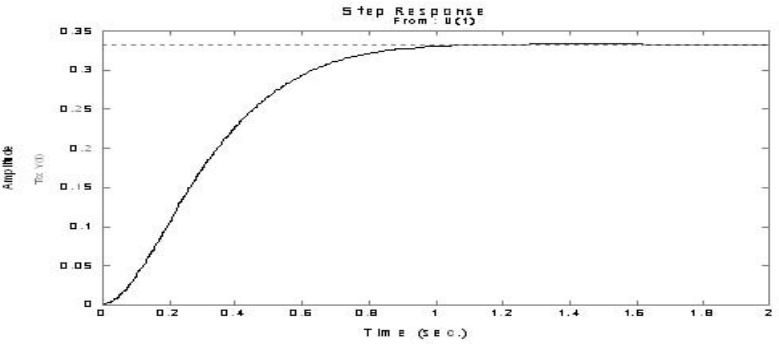


Figure 2.33. Step response with $K_d = 10$

Figure 2.33 illustrates the step response of the system with proportional gain control. Certainly, there is marked improvement in the rise time and also in the final value. The steady-state error has been considerably reduced, but not enough. The main observation is that the rise time is considerably faster than

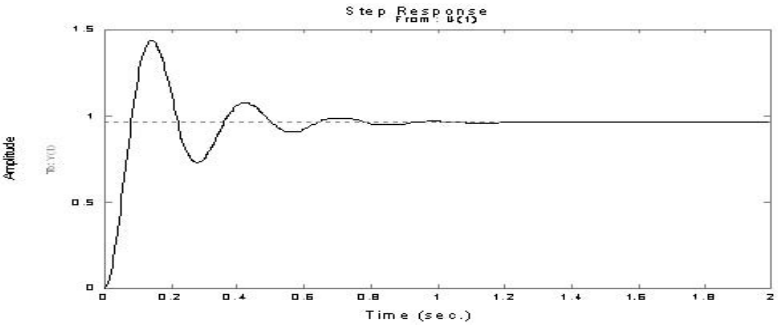


Figure 2.34. Step response with $K_d = 500$

the uncontrolled function. We now take the liberty to increase the proportional gain, say to 500, and try the simulation. Note that our objective is to bring the system response as close to the step input value as possible, in this case 1.0, as fast as we can without causing much overshoot. Figure 2.34 illustrates the effect of increasing the proportional gain. The overshoot is definitely unsatisfactory. It must be reduced to something less than 5%. As far as the settling time is concerned, in this example we are really not concerned because the system we are dealing with has no real-world connection and as such, all we can say is that the performance with control is better than without control. We will see later on, in examples that have a physical meaning, that settling time becomes a very important criteria in the design of controllers. For now, it is important to realize what the proportional, derivative, and integral parameters can do to improve the system response.

We proceed further to examine what we need to do to bring the overshoot down to some tolerable value, typically to less than 5% as we suggested earlier. For this, we need to add derivative control into the system. The following MATLAB code illustrates how this can be done:

```
kp=500; %Proportional gain
kd=10; %Derivative gain
numc=[kd kp] %Define the numerator polynomial (sKd+Kp)
sysopenloop=tf(numc,densys); %Open-loop TF
sysfeedback=[1]; %Unity feedback
sysclosedloop=feedback(sysopenloop,sysfeedback); %Closed-loop TF
step(sysclosedloop,0:0.001:2.0); %Step response
```

As shown in Figure 2.35, the derivative control parameter has most definitely provided damping effect and shows marked improvement in the overshoot compared to the previous case with no derivative control. We also see that the system has settled down more rapidly. So our conclusion is that we can further increase the derivative control so that the response can be brought closer to a critically damped response. Let us try one more simulation in which we increase the derivative gain to 100. The result of this is illustrated in Figure 2.36.

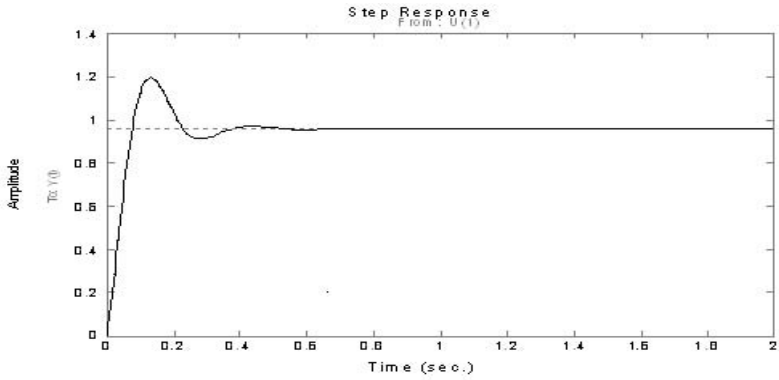


Figure 2.35. Step response with $K_p = 500$ and $K_d = 10$

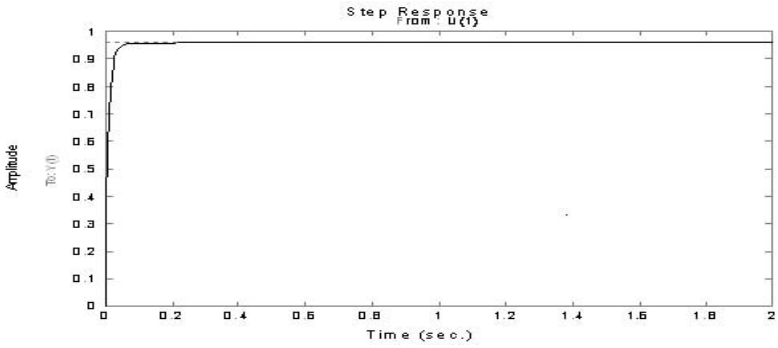


Figure 2.36. Step response with $K_p = 500$ and $K_d = 100$

There is no question that the response is critically damped. However, the steady-state error is not within the acceptable tolerance of being less than 2%. The answer lies in providing an appropriate amount of integral control that will bring the error to within the acceptable tolerance. We can implement integral control action as illustrated in the following MATLAB code:

```
kp=500; %Proportional gain
ki=1; %Integral gain
kd=100; %Derivative gain
numc=[kd kp ki] %Define numerator polynomial of PID controller
denc=[1 0] %Define denominator polynomial of PID controller
den=conv(denc,densys);
%Convolve the denominator polynomials of system and controller
sysopenloop=tf(numc,den); %Obtain open-loop TF
sysfeedback=[1]; %Feedback TF
```

```
sysclosedloop=feedback(sysopenloop,sysfeedback); %Closed-loop TF
step(sysclosedloop,0:0.001:2.0); %Step response
```

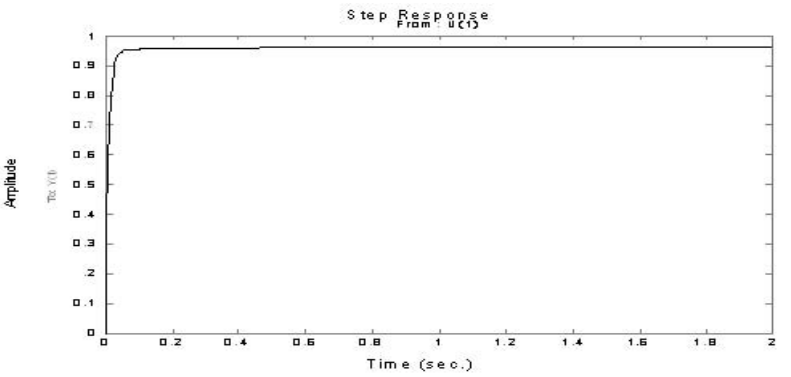


Figure 2.37. Step response with $K_p = 500$, $K_i = 1.0$, and $K_d = 100$

As you can see in Figure 2.37, the addition of a small integral gain, in this case $K_i = 1.0$, has practically no effect in improving the steady-state error. Note that it would be wise to create a bit of an overshoot and then adjust the steady-state error. This is where one needs to perform some trial and error, with some practical knowledge of the limitations of components that will be involved. Saturation and other limiting factors dictate how these parameter variations can be made to obtain satisfactory control performance. In this example, however, we illustrate the systematic nature of the design and the criteria that have to be met in order to build successful controllers.

Proceeding further, we increase the proportional gain and decrease the derivative gain to obtain an overshoot within desirable criteria. We also increase the

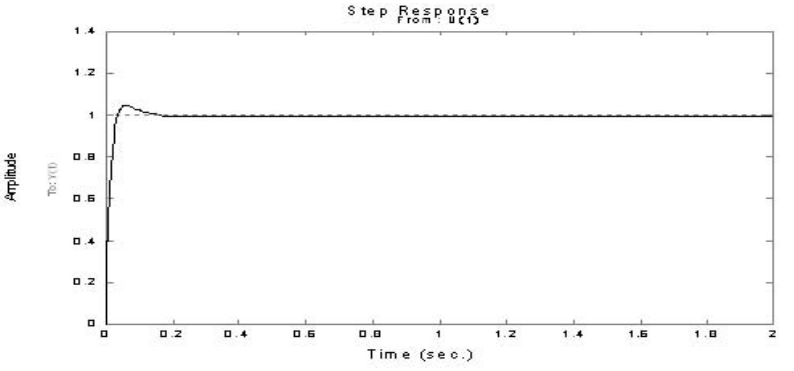


Figure 2.38. Step response with $K_p = 1400$, $K_i = 1000$, and $K_d = 75$

integral gain so that the steady-state error is within a tolerable margin, typically within 2%. Figure 2.38 illustrates the response with a final set of PID control parameters that adequately meet some typical design performance characteristics.

In this section we have very systematically shown the trial and error approach of developing a PID controller for a plant. It is clear that the controller design is facilitated by the use of a linear model. The main problem with such a design is that any major nonlinearity in the plant makes it impossible to design a controller with this approach. As such, there is a need to consider a model-free approach to designing controllers.

2.7.3 Example: controlling dynamics of a servomotor

A DC servomotor is a commonly used actuator in control systems. It provides rotary motion as well as transitional motion. In this example we demonstrate how a controller can be developed to control the dynamics of the servomotor effectively. We will extend this example in Chapter 4 to incorporate a fuzzy controller.

For a classical controller design, we need a model of the system. The objective in this modeling is to obtain a relationship between the angular position of the rotor and the applied voltage. The electric circuit of the armature and the free-body diagram of the rotor are shown in Figure 2.39.

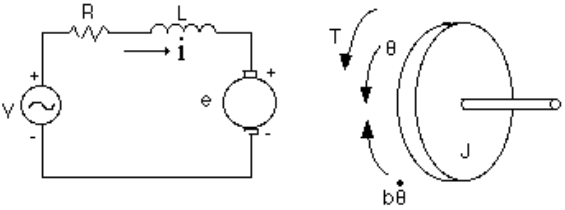


Figure 2.39. DC servomotor

System equations The motor torque T is related to the armature current i by a constant factor K_t as

$$T = K_t i$$

The back electromotive force (emf) e is related to the rotational velocity $\dot{\theta}$ by a constant factor K_e as

$$e = K_e \dot{\theta}$$

In standard international (SI) units, the armature constant K_t, is equal to the motor constant K_e. Therefore, if we let $K = K_t = K_e$, we can write

$$\begin{aligned} T &= Ki \\ e &= K\dot{\theta} \end{aligned}$$

Let J represent the inertia of the wheel and b the damping constant. Also, let L represent the inductance of the armature coil with resistance R. With a voltage V applied to the motor terminals as shown in Figure 2.39, we can write the following coupled electromechanical equations based on Newton's second law and Kirchhoff's law:

$$\begin{aligned} J\frac{d^2\theta}{dt^2} + b\frac{d\theta}{dt} &= T \\ L\frac{di}{dt} + Ri + e &= V \end{aligned}$$

or

$$\begin{aligned} J\frac{d^2\theta}{dt^2} + b\frac{d\theta}{dt} &= Ki \\ L\frac{di}{dt} + Ri + K\frac{d\theta}{dt} &= V \end{aligned}$$

Transfer function The coupled electromechanical equations form the basis for obtaining the input-output relationship of the system. Using Laplace transforms, and with zero initial conditions, the above equations can be expressed as

$$Js^2\Theta(s) + bs\Theta(s) = KI(s)$$

or

$$s\left(Js + b\right)\Theta(s) = KI(s)$$

and

$$LsI(s) + RI(s) + Ks\Theta(s) = V(s)$$

or

$$\left(Ls + R\right)I(s) + Ks\Theta(s) = V(s)$$

Since our objective is to obtain a relationship between rotor angle $\Theta(s)$ and the applied voltage $V(s)$, we eliminate the current $I(s)$. From the first equation we obtain

$$I(s) = \frac{s\left(Js + b\right)}{K}\Theta(s)$$

Substituting this in the second equation and collecting terms, we get

$$\left(\left(Ls + R\right)\frac{s\left(Js + b\right)}{K} + Ks\right)\Theta(s) = V(s)$$

From this we obtain

$$\frac{\Theta(s)}{V(s)} = \frac{1}{\left[\left(Ls + R\right)\frac{s(Js+b)}{K} + Ks\right]} = \frac{K}{[LJs^3 + s^2(RJ + bL) + s(bR + K^2)]}$$

Design specifications Since the DC motor is being used as an actuator, we wish to position the rotor very precisely. Therefore the steady-state error of the motor angular position should be zero. We will also require the steady-state error due to a disturbance to be zero as well. The other performance requirement is that the motor reach its desired position very quickly. In this case, let us specify a settling time requirement of 40 milliseconds. We also want to have an overshoot of less than 5%.

System parameters For our example, let us assume the following parameters for the electromechanical system:

- moment of inertia of the rotor: $J = 3.0 * 10^{-6}\,\mathrm{kg\,m^2/\,s^2}$
- damping ratio of the mechanical system: $b = 3.5 * 10^{-6}\,\mathrm{N\,m/\,s}$
- electromotive force constant: $K = K_e = K_t = 0.03\,\mathrm{N\,m/\,A}$
- electric resistance: $R = 4\,\Omega$
- electric inductance: $L = 2.0 * 10^{-6}\,\mathrm{H}$

Step response We first simulate the open-loop response of this system to determine its behavior to a step input. For this, the MATLAB *m-file* can be set up as follows:

```
J=3.0E-6; %Inertia constant
b=3.5E-6; %Damping constant
K=0.03; %Motor constant
R=4; %Armature resistance
L=2.0E-6; %Armature inductance
num=K; %Transfer function numerator
den=[(J*L) ((J*R)+(L*b)) ((b*R)+K^2) 0];
%Transfer function denominator
sys1=tf(num,den); %MATLAB function "tf" establishes the transfer function
step(sys1,0:0.001:0.2); %Step response with plot
```

As illustrated in Figure 2.40, the rotor position continues to grow when a 1.0 volt step input is applied to the motor terminals. The open-loop response clearly shows the need for a feedback that will stabilize the system for a step input.

Considering a unity feedback system, the closed-loop transfer function of the system can be developed as follows. Let $G_p(s)$ represent the open-loop transfer function. The closed-loop transfer function then becomes $\frac{G_p(s)}{1+G_p(s)}$. This transfer function can be obtained by using MATLAB functions as illustrated below.

```
sys2=1 %Unity feedback transfer function
sys=feedback(sys1,sys2);
%Function "feedback" gives closed-loop transfer function
```

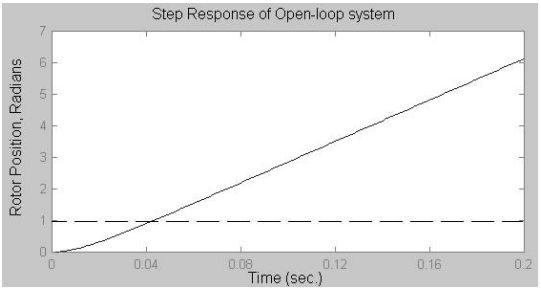


Figure 2.40. Open-loop response

The step response in Figure 2.41 is then obtained by inserting the following MATLAB statement and running the *m-file*.

```
step(sys,0:0.001:0.2); %Step response of closed-loop system with plot;
```

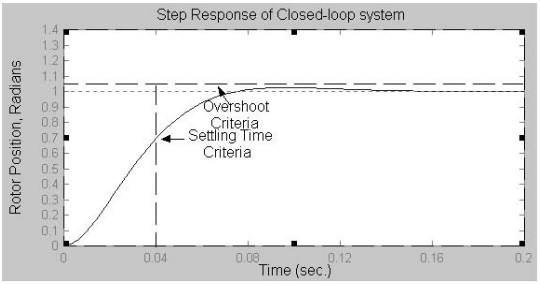


Figure 2.41. Closed-loop response

From the closed-loop response, we observe that the settling time criteria is not satisfied, although the overshoot is within the tolerable limit of less than 5%.

To improve this, we choose the control structure illustrated in Figure 2.42 and we choose the standard PID transfer function for the controller, namely,

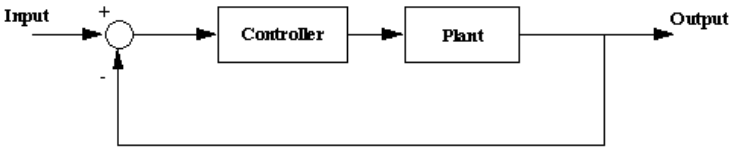


Figure 2.42. Control structure

$$G_c(s) = K_P + \frac{K_I}{s} + K_D s = \frac{K_D s^2 + K_P s + K_I}{s}$$

The objective in the controller design is to select the appropriate parameters for K_P, K_I, and K_D to satisfy all design criteria. Once again, we follow a systematic approach to selecting the parameters. It is important to note that in most cases a PI or PD controller may suffice, and there is really no need for the development of a PID controller. As we will see in this example, a PD controller could adequately bring the system response to within tolerable limits. We do, however, show how a full PID controller may be implemented.

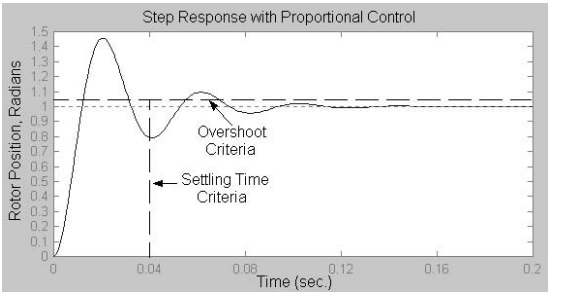


Figure 2.43. Response with proportional gain

To decrease the settling time, we need to add a proportional gain. The proportional gain increases the rate of rise in the system response. If we add only a proportional gain $K_P > 0$, with $K_I = K_D = 0$, then $G_c(s) = K_P$. We can now set up the *m-file* and proceed with MATLAB simulations as follows:

```
%Add proportional control
Kp=10; %This is an arbitrary first choice
numcf=[Kp]; %Numerator of controller transfer function
dencf=[1]; %Denominator of controller transfer function
numf=conv(numcf,num);
%Convolve the numerators of the controller and plant
denf=conv(dencf,den);
%Convolve the denominators of the controller and plant
sysf=tf(numf,denf);
%Form the forward transfer function for controller and plant
sys=feedback(sysf,sys2); %Obtain the overall closed-loop transfer function
step(sys,0:0.001:0.2); %Obtain the step response
```

Notice in Figure 2.43 that the addition of a proportional gain, while sharply increasing the rise time, causes significant overshoot in the response and still does not meet the settling time criteria. Note that the choice of proportional gain is arbitrary. Large values cause excessive oscillation, and small values do not allow the settling time to be met. Some trial and error solutions are needed to obtain what may be considered satisfactory. This is illustrated in

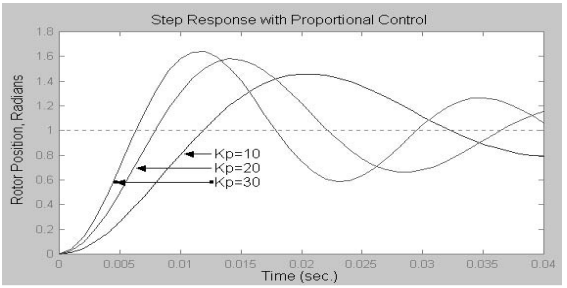


Figure 2.44. System response

Figure 2.44, where the system response is examined for different values of the proportional gain. From the response shown in Figure 2.44, we observe that $K_P = 30$ provides sufficiently good rate of rise so that an appropriate choice of derivative gain may be chosen to reduce the overshoot and thereby improve the settling time. We now examine the system behavior for various values of

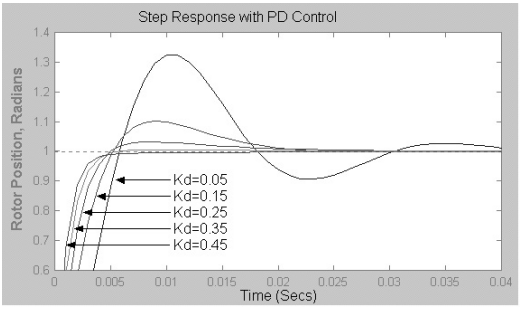


Figure 2.45. System response

derivative gain K_D, with $K_P = 30$. It is clear that for $K_D = 0.35$ the overshoot is negligible, and the settling time is well within the desired specification of 40 milliseconds. Also, the steady-state error is nearly zero. Hence, a PD controller would work well for this example.

Note in Figure 2.45 that for $K_P = 30$ and $K_D = 0.25$, the overshoot is within the 5% tolerance. Suppose we decide to choose these parameters and include integral control to ensure near-zero steady-state error. We could choose K_I in the range of $1 - 100$ and still be within the design specifications. A MATLAB simulation of the final PID controlled system response is illustrated in Figure 2.46.

In summary, the design of a PID controller is iterative in nature. There is no unique solution for the PID parameters. The selection of parameters may however be governed by hardware limitations. A systematic approach can yield

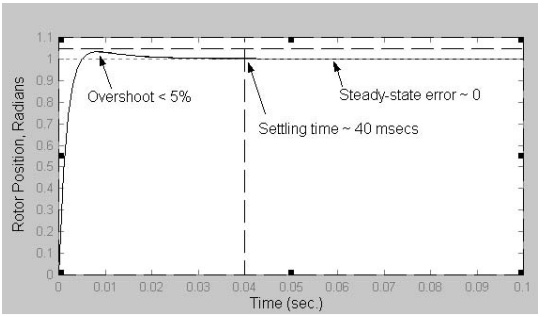


Figure 2.46. System response

a satisfactory controller design. The basic steps in developing PID controllers can be summarized as follows:

- Obtain the open-loop and closed-loop response of the plant and determine what needs to be improved.
- Add proportional control $K_P > 0$ to improve the rise time.
- Add derivative control $K_D > 0$ to improve the overshoot.
- Add integral control to eliminate the steady-state error $K_I > 0$.
- Adjust each of K_P, K_I, and K_D until you obtain a desired overall response.

2.8 Nonlinear control systems

Any system for which the superposition principle does not apply is said to be **nonlinear**. In this case, there is no possibility of generalizing from the response for any class of inputs to the response for any other input. This constitutes a fundamental and important difficulty that necessarily requires any study of nonlinear systems to be quite specific. One can attempt to calculate the response for a specific case of initial conditions and input, but make very little inference based on this result regarding the response characteristics in other cases.

Despite analytical difficulties, one has no choice but to attempt to deal in some way with nonlinear systems, because they occupy a very important place in any practical system. Most linear systems can be thought of as piecewise linear approximations of a nonlinear system. The inverted pendulum example described at the beginning of this chapter is one such example. In some cases the approximation may be very good, but most physical variables, if allowed to take on large values, will eventually go out of their range of reasonable linearity. Most drive systems such as *electrical* and *hydraulic actuators* can only be thought of as being linear over small ranges. *Gas jets*, for example, have no linear range at all, and to achieve minimum-time control, an ON-OFF (bang-bang or relay)

type controller is used in missile and spacecraft control systems. Other types of nonlinearities have to be dealt with in the controller design as well.

There is no closed-form solution summarizing the response characteristics of nonlinear systems. These systems display a variety of behaviors that have to be studied quite specifically using some iterative solution technique. Typically, the solution of nonlinear systems requires simulation of the dynamic behavior for a variety of inputs and initial conditions. The most obvious departure from linear system behavior is the dependence of the response on the amplitude of the excitation. This excitation can be either initial conditions or the forcing input, or both. A common situation is a system that responds to some small initial conditions by returning to rest in a well-behaved stable manner, while diverging in an unstable manner for some other initial conditions. This type of behavior is classified as **saddle-point behavior**. In some cases, the response to certain initial conditions may lead to continuous oscillation, the characteristics of which are a property of the system. Such oscillations give rise to two classes of systems, namely, systems that exhibit **limit cycle behavior** and systems that are **chaotic**. A limit cycle is the trajectory of the system in its state space where the trajectory closes in on itself and will not result in a steady state. A chaotic system on the other hand, is a system for which trajectories in the state space are unique and never follow the same path twice. This phenomenon is not possible in linear systems.

2.9 Linearization

A study and design of controllers for nonlinear systems can be greatly simplified by approximating it about a desired operating point by a linear system. The resulting linear system allows one to say a great deal about the performance of the nonlinear system for small departures around the operating point. Any response that carries variables through a range that exceeds the limits of reasonable linear approximation does not reflect the behavior of the nonlinear system. This requires repeated linearizations about new operating points, and the resulting solutions being "patched" together. In fact, this technique of "patching" has significant connotation in the ability of fuzzy logic-based systems being "universal approximators." A discussion of universal approximation is provided in Chapter 4 on fuzzy control. Consider a nonlinear system represented by the equation

$$\dot{x}(t) = f\left(x(t), u(t)\right) \tag{2.41}$$

Linearization can be performed by expanding the nonlinear equations into a Taylor series about an operating point and neglecting all terms higher than first-order terms. Let the nominal operating point be denoted $x_0(t)$, which corresponds to the nominal input $u_0(t)$ and some fixed initial states. Expanding Equation (2.41) into a Taylor series about $x(t) = x_0(t)$, and neglecting all terms

higher than first-order terms gives

$$\dot{x}_i(t) = f_i[\mathbf{x}_0(t), \mathbf{u}_0(t)] + \sum_{j=1}^{n} \left.\frac{\partial f_i}{\partial x_j}\right|_{x_0,u_0} (x_j - x_{0j}) + \sum_{j=1}^{n} \left.\frac{\partial f_i}{\partial u_j}\right|_{x_0,u_0} (u_j - u_{0j}) \quad (2.42)$$

where, $i = 1, 2, ..., n$.

Now, if we let

$$\begin{aligned} \triangle x_i &= x_i - x_{0i} \\ \triangle u_i &= u_i - u_{0i} \end{aligned}$$

then

$$\triangle \dot{x}_i = \dot{x}_i - \dot{x}_{0i}$$

Since $\dot{x}_{0i} = f_i[\mathbf{x}_0(t), \mathbf{u}_0(t)]$, Equation (2.42) can be written as

$$\triangle \dot{x}_i = \sum_{j=1}^{n} \left.\frac{\partial f_i}{\partial x_j}\right|_{x_0,u_0} (\triangle x_j) + \sum_{j=1}^{n} \left.\frac{\partial f_i}{\partial u_j}\right|_{x_0,u_0} (\triangle u_j) \quad (2.43)$$

Equation (2.43) can be expressed in vector-matrix form as

$$\triangle \dot{\mathbf{x}} = \mathbf{A}^* \triangle \mathbf{x} + \mathbf{B}^* \triangle \mathbf{u} \quad (2.44)$$

where

$$\mathbf{A}^* = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad \mathbf{B}^* = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \cdots & \frac{\partial f_2}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \frac{\partial f_n}{\partial u_2} & \cdots & \frac{\partial f_n}{\partial u_n} \end{bmatrix} \quad (2.45)$$

Both $\mathbf{A}^*$ and $\mathbf{B}^*$ are evaluated at the nominal operating point. In general, however, Equation (2.44), although linear, may contain time varying elements.

Example 2.8 (Linearization) Suppose we wish to linearize the following state equations of a nonlinear system:

$$\dot{x}_1(t) = \frac{-1}{x_2^2(t)} \quad (2.46)$$

$$\dot{x}_2(t) = x_1(t)\, u(t) \quad (2.47)$$

These equations are to be linearized about the nominal trajectory $[x_{01}(t), x_{02}(t)]$ *which* is the solution to the equations with initial conditions $x_{01}(0) = x_{02}(0) = 1$ and input $u(t) = 0$.

Integrating Equation (2.47) with respect to time t under the specified initial conditions, we get

$$x_2(t) = x_2(0) = 1 \quad (2.48)$$

Then Equation (2.46) gives

$$x_1(t) = -t + 1 \tag{2.49}$$

Therefore, the nominal trajectory for which Equations (2.46) and (2.47) are to be linearized is described by

$$x_{01}(t) = -t + 1 \tag{2.50}$$

$$x_{02}(t) = 1 \tag{2.51}$$

Now evaluating the coefficients of Equation (2.45), we get

$$\frac{\partial f_1(t)}{\partial x_1(t)} = 0 \qquad \frac{\partial f_1(t)}{\partial x_2(t)} = \frac{2}{x_2^3(t)} \qquad \frac{\partial f_1(t)}{\partial u(t)} = 0$$

$$\frac{\partial f_2(t)}{\partial x_1(t)} = u(t) \qquad \frac{\partial f_2(t)}{\partial x_2(t)} = 0 \qquad \frac{\partial f_2(t)}{\partial u(t)} = x_1(t)$$

Using Equation (2.43), we get

$$\triangle \dot{x}_1(t) = \frac{2}{x_{02}^3(t)} \triangle x_2(t) \tag{2.52}$$

$$\triangle \dot{x}_2(t) = u_0(t) \triangle x_1(t) + x_{01}(t) \triangle u(t) \tag{2.53}$$

Substituting Equations (2.50) and (2.51) into (2.52) and (2.53), we get

$$\begin{bmatrix} \triangle \dot{x}_1(t) \\ \triangle \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \triangle x_1(t) \\ \triangle x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1-t \end{bmatrix} \triangle u(t) \tag{2.54}$$

The set of Equations (2.54) represents linear state equations with time-varying coefficients.

2.10 Exercises and projects

1. Show that the following systems are either linear or nonlinear, and either time-varying or time-invariant.

   (a) $y(t) = v(t) \frac{d}{dt}(v(t))$

   (b) $\xrightarrow{v(t)} \boxed{N_1} \xrightarrow{q(t)} \boxed{N_2} \xrightarrow{y(t)}$ where N_1 and N_2 are linear systems connected in cascade.

2. Prove or disprove the following statements.

   (a) In a linear system, if the response to $v(t)$ is $y(t)$, then the response to $\mathrm{Re}(v(t))$ is $\mathrm{Re}(y(t))$ where $\mathrm{Re}(x)$ denotes the real part of the complex number x.

   (b) In a linear system, if the response to $v(t)$ is $y(t)$, then the response to $\frac{d}{dt}(v(t))$ is $\frac{d}{dt}(y(t))$.

3. Each of the following is a linear time-invariant system.

(i) $\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 2 & 2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix} u \qquad y = \begin{bmatrix}2 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$

(ii) $\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 0 & -2 & -3\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}0\\ 1\\ 1\end{bmatrix} u \qquad y = \begin{bmatrix}1 & 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$

(iii) $\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 1\\ 0 & -2 & -1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}1\\ 0\\ 1\end{bmatrix} u \qquad y = \begin{bmatrix}0 & 1 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$

(iv) $\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix} = \begin{bmatrix}-1 & 0 & 0\\ 0 & -1 & 0\\ 0 & -2 & -2\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \begin{bmatrix}0\\ 1\\ 1\end{bmatrix} u \qquad y = \begin{bmatrix}1 & 1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$

   (a) Explain why each system is or is not fully controllable.

   (b) Explain why each system is or is not fully observable.

   (c) Find the transfer function for each system.

   (d) Explain why each system is or is not stable.

4. For each of the open-loop transfer functions of linear time-invariant systems, specified below, we are required to obtain state-variable feedback controllers. The corresponding desired roots (characteristic equation) are specified for each system.

(i) $G_1(s) = \dfrac{(s+2)}{(s+3)(s+7)}$; Desired roots: $\{-5,\ -8\}$

(ii) $G_2(s) = \dfrac{10}{s(s+1)(s+5)}$; Desired roots: $\{-0.708 \pm j0.706,\ -50\}$

(iii) $G_3(s) = \dfrac{(s^2+2)}{(s+3)(s+4)(s+5)}$; Desired characteristic equation:

$$s^3 + 8s^2 + 22s + 24 = 0$$

(iv) $G_4(s) = \dfrac{(s^2+s+1)}{(s^3+8s^2+22s+24)}$; Desired characteristic equation:

$$s^3 + 15s^2 + 71s + 105 = 0$$

   (a) For each system obtain the step input open-loop response using MATLAB.

   (b) For each system with state-variable feedback, obtain the step input response using MATLAB.

5. Explain why $C\left(sI - A\right)^{-1} B + E = \dfrac{\hat{y}\left(s\right)}{\hat{u}\left(s\right)}$ as claimed in Equations 2.39 and 2.40.

6. A system is described by the following differential equation

$$\frac{d^2x}{dt^2} + 2\frac{dx}{dt} + 3x = 1$$

with the initial conditions $x(0) = 1$ and $\dot{x}(0) = -1$. Show a block diagram of the system, giving its transfer function and all pertinent inputs and outputs.

7. Develop the state model (state and output equations) for the system below.

$$R(s) \longrightarrow \boxed{\frac{5s+1}{s^4+2s^3+s^2+5s+10}} \longrightarrow C(s)$$

8. A system is defined as

$$\frac{d^2x}{dt^2} + 12\frac{dx}{dt} + 30x = f(x)$$

Linearize the system for the following functions $f(x)$.

(a) $f(x) = \sin x$ for $x = 0$ (b) $f(x) = \sin x$ for $x = \pi$

(c) $f(x) = e^{-x}$ for $x \approx 0$

9. A first-order system is modeled by the state and output equations as

$$\begin{aligned} \frac{dx(t)}{dt} &= -3x(t) + 4u(t) \\ y(t) &= x(t) \end{aligned}$$

(a) Find the Laplace transform of the set of equations and obtain the transfer function.

(b) If the input $u(t)$ is a unit step function, with $x(0) = 0$, find $y(t)$, $t > 0$.

(c) If the input $u(t)$ is a unit step function, with $x(0) = -1$, find $y(t)$, $t > 0$.

(d) Obtain a MATLAB solution to verify your results.

10. Given the state equations

$$\begin{aligned} \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} &= \begin{bmatrix} 0 & 2 \\ -1 & -3 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \end{aligned}$$

(a) Find the Laplace transform of the set of equations and obtain the transfer function.

(b) If the input $u(t)$ is a unit step function, with $x_1(0) = x_2(0) = 0$, find $y(t)$, $t > 0$.

(c) If the input $u(t)$ is a unit step function, with $x_1(0) = x_2(0) = -1$, find $y(t)$, $t > 0$.

(d) Obtain a MATLAB solution to verify your results.

11. For each second-order system given below, obtain a complete analytical solution $y(t)$, $t > 0$ when $x_1(0) = 1$ and $x_2(0) = 0$. Assume the input $u(t) = u_1(t) = u_2(t)$ is a unit step function. Using MATLAB, verify your solutions in each case.

(a) $$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} -3 & 2 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u(t)$$
$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

(b) $$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 3 \\ -5 & -8 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix}$$
$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

(c) $$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 4 \\ 0 & -5 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$
$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

12. Consider the following plant models:

(a) $G_{p1}(s) = \dfrac{10}{(s+1)(s+10)}$ (b) $G_{p2}(s) = \dfrac{1}{(s+1)(s-3)}$

(c) $G_{p3}(s) = \dfrac{1}{(s^2+s+1)}$ (d) $G_{p4}(s) = \dfrac{1}{s(s+1)}$

Making reasonable assumptions for settling time and steady-state error criteria, for each plant model derive a suitable set of PID control parameters. Use MATLAB to simulate the behavior of the system. Assume a tolerable overshoot of less than 5% in all cases.

13. Synthesize a PI controller for the plant given by the transfer function

$$G_p(s) = \frac{1}{(s^2+6s+9)}$$

Simulate the dynamics of the plant using MATLAB. State all assumptions.

14. Consider the plant model of a system as follows:

$$G_p(s) = \frac{-\alpha s+1}{(s^2+3s+2)}$$

Synthesize a set of PID parameters for different values of α in the interval [0.1; 20]. Discuss your results.

15. For the inverted pendulum problem example in Section 2.1.2, consider the following parameters

$$\begin{aligned} M &= 10.5\,\text{kg} \\ m &= 0.5\,\text{kg} \\ b &= 0.01\,\text{N/ m/ s} \\ l &= 0.6\,\text{m} \\ I &= 0.016\,\text{kg m}^2 \end{aligned}$$

Consider the effects of changing proportional gain K_P over a range of $0-1000$, the derivative gain K_D over a range of $0-500$, and the integral gain K_I over the range of $0-10$. Obtain a suitable set of PID parameters that will meet the standard criteria of less than 5% overshoot and a two second settling time.

16. For the inverted pendulum problem, suppose that it is necessary for the cart to return to the same position or to a desired position. Include cart control in your model and obtain simulations using Simulink.

17. **Project**: For the ardent student, modeling the ancient control system developed by Hero should be a challenge. While there are no specific guidelines that can be given, commonsense and reasonableness account for obtaining the mathematical model of the system. Some hints in modeling, however, can be useful. A simple lag, for example, is given by

$$G_1(s) = \frac{1}{s+T_1}$$

Note that the inverse Laplace transform yields

$$g_1(t) = e^{-T_1 t}u(t)$$

This should provide the insight into choosing the appropriate time constants, like T_1, so that fast and slow responding actions can be appropriately modeled. The goal, of course, is to design a controller that will perform the opening and closing of the temple doors.

Chapter 3

FUZZY LOGIC FOR CONTROL

In this chapter, we set forth the basic mathematical ideas used in fuzzy control. These ideas are illustrated here with simple examples. Their applications will be expanded upon in later chapters.

3.1 Fuzziness and linguistic rules

There is an inherent impreciseness present in our natural language when we describe phenomena that do not have sharply defined boundaries. Such statements as "Mary is smart" and "Martha is young" are simple examples. Fuzzy sets are mathematical objects modeling this impreciseness.

Our main concern is representing, manipulating, and drawing inferences from such imprecise statements. Fuzzy set theory provides mathematical tools for carrying out approximate reasoning processes when available information is uncertain, incomplete, imprecise, or vague. By using the concept of degrees of membership to give a mathematical definition of fuzzy sets, we increase the number of circumstances encountered in human reasoning that can be subjected to scientific investigation.

Humans do many things that can be classified as control. Examples include riding a bicycle, hitting a ball with a bat, and kicking a football through the goalposts. How do we do these things? We do not have the benefit of precise measurements, or a system of differential equations, to tell us how to control our motion, but humans can nevertheless become very skillful at carrying out very complicated tasks. One explanation is that we learn through experience, common sense, and coaching to follow an untold number of basic rules of the form "If...then...":

If the bicycle leans to the right, then turn the wheel to the right.
If the ball is coming fast, then swing the bat soon.

If a strong wind is blowing right to left, then aim to the right of the goalposts.

The use of basic rules of this form is the basic idea behind fuzzy control. Linguistic variables such as fast, slow, large, medium, and small are translated into fuzzy sets; mathematical versions of "If...then..." rules are formed by combining these fuzzy sets.

3.2 Fuzzy sets in control

It is easy to express rules in words, but the linguistic notion of fuzziness as illustrated in the rules above needs to be represented in a mathematical way in order to make use of this notion in control theory. How can we do that? The mathematical modeling of fuzzy concepts was first presented by Professor Lotfi Zadeh in 1965 to describe, mathematically, classes of objects that do not have precisely defined criteria of membership. His contention is that meaning in natural language is a matter of degree.

Zadeh gave the examples, "the class of all beautiful women" and "the class of all tall men." The notion of "tall" can be depicted by a graph such as Figure 3.1, where the x-axis represents height in centimeters, and the y-axis represents the degree, on a scale of 0 to 1, of the tallness attributed to that height. Of

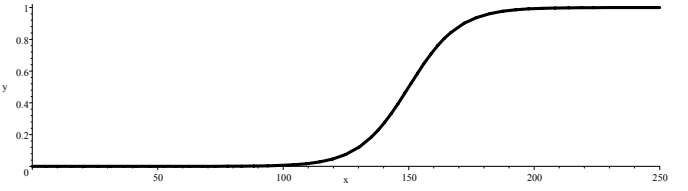


Figure 3.1. "Tallness" of height in centimeters

course, the scaling of this function depends on the context, but the shape is descriptive in a fairly general setting.

Before defining a fuzzy subset mathematically, we first look at ordinary subsets in a special way that will allow us to expand the notion of subset to that of fuzzy subset. An ordinary subset A of a set X can be identified with a function $X \to \{0,1\}$ from X to the 2-element set $\{0,1\}$, namely

$$A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

This function is called the **characteristic function** or **indicator function** of the fuzzy set A. If A is the set of real numbers equal to or bigger than 10 and less than or equal to 40, the set A would be depicted as in Figure 3.2. In contrast, elements of fuzzy subsets of X can have varying degrees of membership from 0 to 1.

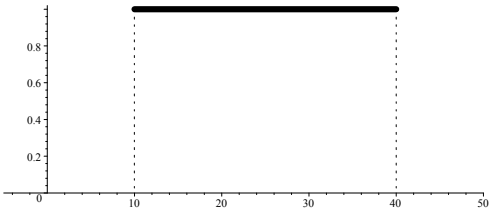


Figure 3.2. $A = [10, 40]$

Definition 3.1 *A **fuzzy subset** A of a set X is a function $A : X \to [0, 1]$ from X to the unit interval $[0, 1]$.*

The value of $A(x)$ is thought of as the degree of membership of x in A. This function is sometimes called the **membership function** of A. In the special case that the membership function takes on only the values 0 and 1, A is called an **ordinary** or **crisp subset** of X and its membership function coincides with its characteristic function. The set X is sometimes called the **universe of discourse**. A fuzzy subset is often referred to simply as a **fuzzy set**.

Other notation in common use for the membership function of the fuzzy subset A of X includes $\mu_A : X \to [0, 1]$ and sometimes $\int_{x \in X} \mu_A(x)/x$ or, if the domain is discrete, $\sum_{x \in X} \mu_A(x)/x$. These more complicated notations do not convey additional information, and we do not make a notational distinction between a fuzzy subset A and its membership function $A : X \to [0, 1]$.

The **support** of a function $A : X \to [0, 1]$ is the set

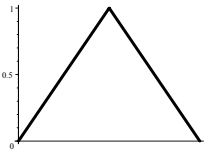
$$\text{supp}(A) = \{x \in X \mid A(x) \neq 0\}$$

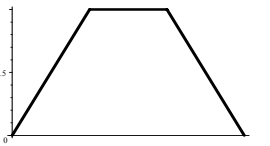
For most purposes, it is not critical whether X is the support of A or some set larger than the support. Thus, given a collection of fuzzy subsets of the real numbers, for example, we may assume that they share the same universe of discourse by taking X to be the union of the supports of the fuzzy sets. If two fuzzy sets have the same support and take the same values on their support, we generally need not distinguish between them.

A fuzzy set $A : X \to [0, 1]$ is **normal** if there is an $x \in X$ such that $A(x) = 1$. A fuzzy set $A : \mathbb{R} \to [0, 1]$ is **convex** if given $x \leq y \leq z$, it must be true that $f(y) \geq f(x) \wedge f(z)$. A fuzzy set $A : \mathbb{R} \to [0, 1]$ with finite support that is both normal and convex is often called a **fuzzy number**. Most fuzzy sets used in control are both normal and convex.

Since data is generally numerical, the universe of discourse is most often an interval of real numbers, or in practice, a finite set of real numbers. The shape of a membership function depends on the notion the set is intended to describe and on the particular application involved. The membership functions most commonly used in control theory are triangular, trapezoidal, Gaussian, and sigmoidal Z- and S-functions, as depicted in the following figures.

Triangles and **trapezoids**, which are piecewise-linear functions, are often used in applications. Graphical representations and operations with these fuzzy sets are very simple. Also, they can be constructed easily on the basis of little information.





Triangular Trapezoidal

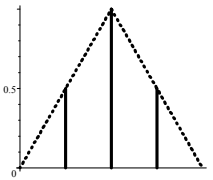
The triangular function A with endpoints $(a, 0)$ and $(b, 0)$, and high point (c, α) is defined by

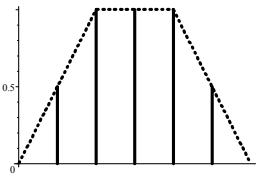
$$A(x) = \begin{cases} \alpha\left(\dfrac{x-a}{c-a}\right) & \text{if } a \le x \le c \\ \alpha\left(\dfrac{x-b}{c-b}\right) & \text{if } c \le x \le b \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The trapezoidal function B with endpoints $(a, 0)$ and $(b, 0)$, and high points (c, α) and (d, α) is defined by

$$A(x) = \begin{cases} \alpha\left(\dfrac{x-a}{c-a}\right) & \text{if } a \le x \le c \\ \alpha & \text{if } c \le x \le d \\ \alpha\left(\dfrac{x-b}{d-b}\right) & \text{if } d \le x \le b \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

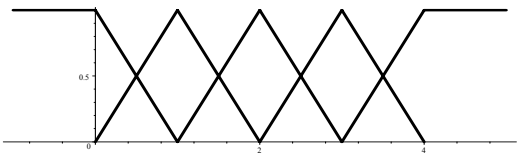
When the domain is finite, these fuzzy sets can be depicted as follows:





Triangular Trapezoidal

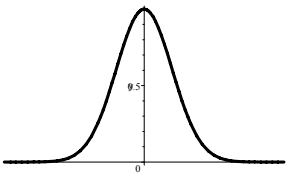
Several fuzzy sets are often depicted in the same plot, as follows:



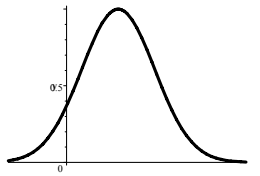
The **Gaussian functions**, the familiar bell-shaped curve, are of the form

$$A(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$$

These are related to the well-known normal or Gaussian distributions in probability and have useful mathematical properties.



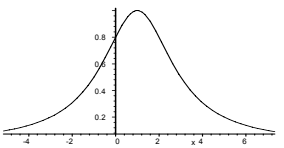
Gaussian $e^{-\frac{x^2}{2}}$



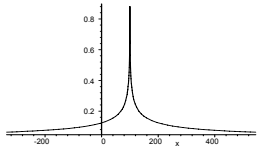
Gaussian $e^{-\frac{(x-5)^2}{25}}$

The parameters c and σ determine the center and the shape of the curve, respectively. The values $c = 0$ and $\sigma = 1$ define the *standard Gaussian membership function* $e^{-\frac{x^2}{2}}$, centered at $c = 0$, and with area under the curve equal to $\sqrt{2\pi}$. This is the Gaussian curve depicted on the left above.

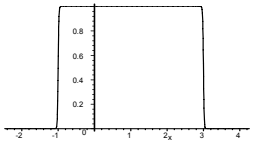
A Cauchy function, or **generalized bell curve**, is given by functions of the form $A(x) = 1/\left(1 + \left|\frac{x-c}{a}\right|^{2b}\right)$. The parameter c determines the center of the curve, and a and b determine its shape.



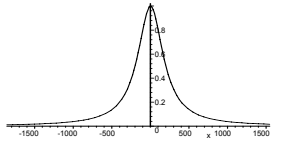
$1\Big/\left(1 + \left|\frac{x-1}{2}\right|^2\right)$



$1\Big/\left(1 + \left|\frac{x-100}{2}\right|^{1/2}\right)$



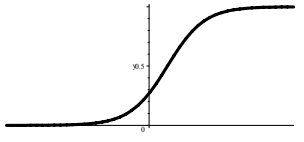
$1\Big/\left(1 + \left|\frac{x-1}{2}\right|^{200}\right)$



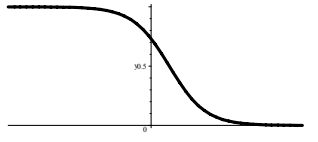
$1\Big/\left(1 + \left|\frac{x-1}{200}\right|^2\right)$

The S- and Z-functions are **sigmoidal functions** of the form

$$A(x) = \frac{1}{1 + e^{-(x-m)\sigma}}$$

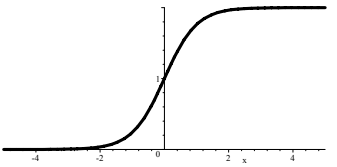


S-function $\frac{1}{1+e^{-x+1}}$

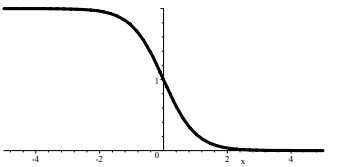


Z-function $\frac{1}{1+e^{x-1}}$

The values of σ determine either increasing or decreasing functions, while the parameter m shifts the function right or left. These same shapes can be achieved with hyperbolic tangent functions since $\frac{1}{2}(1+\tanh x) = \frac{1}{1+e^{2x}}$:

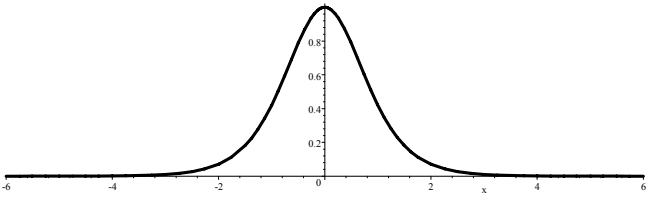


$1+\tanh x$



$1-\tanh x$

The product of two sigmoidal functions is sometimes used



$1-\tanh^2 x$

All of these functions can be useful for different applications.

3.3 Combining fuzzy sets

In fuzzy control theory, where we work with collections of fuzzy subsets, we need useful ways to combine them. These ways of combining should coincide with known methods when the sets in question are ordinary sets. In other words, methods of combining fuzzy sets should generalize common methods for ordinary sets. The various operators used to combine fuzzy sets are called **fuzzy connectives** or **aggregation operators**.

The variety of operators for the aggregation of fuzzy sets can be confusing. If fuzzy set theory is used as a modeling language for real situations or systems, it is not only important that the operators satisfy certain axioms or have certain formal qualities (such as associativity and commutativity), that are certainly of importance, but the operators must also be appropriate models of real-system behavior, and this can normally be proven only by empirical testing. In practice, numerical efficiency in computations can also be an important consideration.

3.3.1 Minimum, maximum, and complement

Ordinary subsets, also known as **crisp** sets, of a set X are often combined or negated via **intersection** (AND), **union** (OR), and **complement** (NOT):

$$\begin{array}{ll} \text{AND:} & A \cap B = \{x \in X : x \in A \text{ and } x \in B\} \text{ (intersection)} \\ \text{OR:} & A \cup B = \{x \in X : x \in A \text{ or } x \in B\} \text{ (union)} \\ \text{NOT:} & X - A = \{x \in X : x \notin A\} \text{ (complement)} \end{array} \tag{3.3}$$

These operations are determined by their **characteristic functions**. The characteristic functions for intersection, union, and complement are

$$\begin{aligned}(A\cap B)(x) &= \begin{cases}1 & \text{if } x\in A \text{ and } x\in B\\ 0 & \text{if } x\notin A \text{ or } x\notin B\end{cases}\\ (A\cup B)(x) &= \begin{cases}1 & \text{if } x\in A \text{ or } x\in B\\ 0 & \text{if } x\notin A \text{ and } x\notin B\end{cases}\\ (X-A)(x) &= \begin{cases}1 & \text{if } x\notin A\\ 0 & \text{if } x\in A\end{cases}\end{aligned} \tag{3.4}$$

For these characteristic functions representing ordinary sets, the following are satisfied. You will be asked to prove the facts in Equations 3.5 in the exercises.

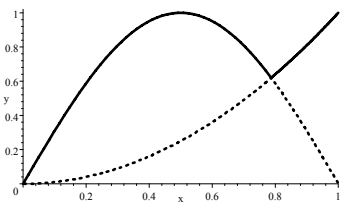
$$\begin{aligned}(A\cap B)(x) &= A(x)\wedge B(x) = A(x)\,B(x)\\ &= \max\{A(x)+B(x)-1,0\}\\ (A\cup B)(x) &= A(x)\vee B(x) = A(x)+B(x)-A(x)\,B(x)\\ &= \min\{A(x)+B(x),1\}\\ (X-A)(x) &= 1-A(x) = (1-A(x)^a)^{\frac{1}{a}} = \frac{1-x}{1+(a-1)\,x} \text{ for } a>0\end{aligned} \tag{3.5}$$

Each of the above equations leads to a generalization of AND, OR, or NOT for fuzzy subsets. By far the most common generalizations are the classical fuzzy operators

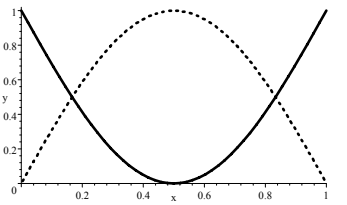
$$\begin{aligned}\text{AND:}&\quad (A\wedge B)(x) = A(x)\wedge B(x) \text{ (minimum or meet)}\\ \text{OR:}&\quad (A\vee B)(x) = A(x)\vee B(x) \text{ (maximum or join)}\\ \text{NOT:}&\quad (\neg A)(x) = 1-A(x) \text{ (complement or negation)}\end{aligned} \tag{3.6}$$

where $A(x)\wedge B(x) = \min\{A(x),B(x)\}$ and $A(x)\vee B(x) = \max\{A(x),B(x)\}$. The membership functions for these fuzzy sets will be denoted by $A\wedge B$ for minimum, $A\vee B$ for maximum, and A' or $\neg A$ for complement.

Example 3.1 Suppose $X=[0,1]$, $A(x)=x^2$, and $B(x)=\sin\pi x$. The sets $A\vee B$, $A\wedge B$, and B' are shown below.



$A\vee B$ (solid), $A\wedge B$ (dotted)



B (dotted), B' (solid)

There is an important relation that exists among these three operations, known as the **De Morgan Laws**.

$$(A\vee B)' = A'\wedge B' \text{ and } (A\wedge B)' = A'\vee B' \tag{3.7}$$

We say that each of these binary operations is **dual** to the other, relative to the complement. You will be asked in the exercises to verify that these laws hold. An equivalent way to view these laws is

$$A \vee B = (A' \wedge B')' \text{ and } A \wedge B = (A' \vee B')'$$

from which you can see that knowing either of the binary operations and the complement completely determines the dual binary operation.

3.3.2 Triangular norms, conorms, and negations

Although minimum, maximum, and complement are the most common operations used for AND, OR, and NOT, there are many other possibilities with fuzzy sets. Some of these are described in the next sections.

Triangular norms Some situations call for a different notion of AND, and in most cases these notions are supplied by *triangular norms*, usually referred to as *t-norms*. The notion of AND suggested in Equations 3.6 — $A(x) \wedge B(x)$ — is an example of a t-norm. At the end of this section, we look at some nonassociative generalizations of AND and OR.

The general definition of a t-norm is the following.

Definition 3.2 *A* ***t-norm*** *is a binary operation* $\circ : [0,1] \times [0,1] \to [0,1]$ *satisfying for all* $x, y, z \in [0,1]$

1. $x \circ y = y \circ x$ ($\circ$ *is commutative*)
2. $x \circ (y \circ z) = (x \circ y) \circ z$ ($\circ$ *is associative*)
3. $x \circ 1 = 1 \circ x = x$ (1 *is an identity*)
4. $y \leq z$ *implies* $x \circ y \leq x \circ z$ ($\circ$ *is increasing in each variable*)

The term "binary operation" simply means that a t-norm is a function of two variables. In the literature, you will often find t-norms written as functions of two variables with the notation $x \circ y = T(x, y)$. Using this notation, the commutative, associative, and identity laws look like

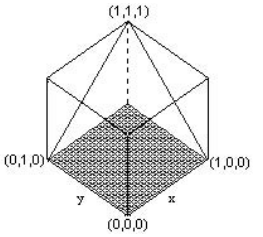
1. $T(x, y) = T(y, x)$
2. $T(x, T(y, z)) = T(T(x, y), z)$
3. $T(x, 1) = T(1, x) = x$

We use the notation $x \circ y$ because it is simpler and because binary notation is commonly used for intersection ($A \cap B$), minimum ($x \wedge y$) and, as in the next example, product (where the symbol is suppressed entirely except when needed for clarity).

The most widely used t-norm, minimum, is also the largest t-norm — that is,

$$x \circ y \leq x \wedge y$$

for any t-norm $\circ$. The smallest t-norm, called the **drastic product**, is not continuous.



$$\text{Drastic product} \qquad x \circ y = \begin{cases} x \wedge y & \text{if} \quad x \vee y = 1 \\ 0 & \text{if} \quad x \vee y < 1 \end{cases}$$

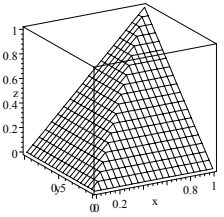
The most common examples of a t-norm, other than minimum, are the **product t-norm**, or **algebraic product**

$$x \circ y = xy$$

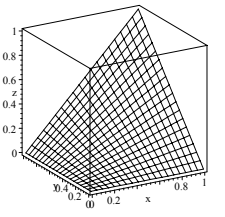
and the **Łukasiewicz**[1] **t-norm** or **bounded product**

$$x \circ y = \max\{x + y - 1, 0\} = (x + y - 1) \vee 0$$

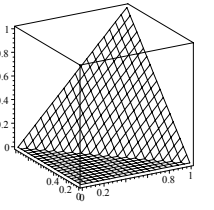
These three t-norms (minimum, algebraic product, and bounded product) are depicted in the following plots:



$x \wedge y$



xy



$\max\{x + y - 1, 0\}$

A t-norm that satisfies $x \circ x = x$ for all x is called **idempotent**. The minimum t-norm is idempotent, and it is the only idempotent t-norm. A continuous t-norm that satisfies $x \circ x < x$ for all $x \neq 0, 1$ is called **Archimedean**. All continuous t-norms we will consider, other than minimum, are Archimedean. The three examples depicted above are basic, in the sense that all other continuous t-norms can be obtained from these three in a straightforward way.

An Archimedean t-norm for which $x \circ x = 0$ only when $x = 0$ is called **strict**. The product t-norm is the prototype for a strict t-norm. Other Archimedean t-norms are **nilpotent**. The Łukasiewicz t-norm is the prototype for a nilpotent t-norm.

[1] Łukasiewicz (1878–1956) was a Polish logician and philosopher.

The following theorem has its roots in the work of the Norwegian mathematician Niels Abel in the early 1800's, long before the idea of fuzzy sets arose. A function $f : [0,1] \to [0,1]$ is an **order isomorphism** if it is continuous and strictly increasing with $f(0) = 0$ and $f(1) = 1$.

Theorem 3.1 *An Archimedean t-norm $\circ$ is strict if and only if there is an order isomorphism $f : [0,1] \to [0,1]$ satisfying the identity*

$$f(x \circ y) = f(x)f(y)$$

or equivalently, such that

$$x \circ y = f^{-1}(f(x)f(y))$$

Another such order isomorphism g satisfies this condition if and only if $f(x) = g(x)^r$ for some $r > 0$.

A similar theorem is true for nilpotent t-norms.

Theorem 3.2 *An Archimedean t-norm $\circ$ is nilpotent if and only if there is an order isomorphism $f : [0,1] \to [0,1]$ satisfying the identity*

$$f(x \circ y) = (f(x) + f(y) - 1) \vee 0$$

or equivalently, such that

$$x \circ y = f^{-1}\left((f(x) + f(y) - 1) \vee 0\right)$$

Another such order isomorphism g satisfies this condition if and only if $f = g$.

Another way to express the situation is: Every strict t-norm is *isomorphic* to the product t-norm and every nilpotent t-norm is *isomorphic* to the Łukasiewicz t-norm. The order isomorphisms f in the theorems above are called **generators** of the resulting t-norms.

A strict t-norm $T(x,y) = x \circ y$ is obtained from the algebraic product and a generator f by

$$x \circ y = f^{-1}\left(f(x)\, f(y)\right)$$

The "additive" generator $F : [0,1] \to [0,\infty]$ for strict t-norms that is commonly referred to in the literature is related to f by $F(x) = -\ln f(x)$. Thus, there is a decreasing function $F : [0,1] \to [0,\infty]$ with

$$x \circ y = F^{-1}\left(F(x) + F(y)\right)$$

A nilpotent t-norm $T(x,y) = x \bullet y$ can be obtained from the bounded product and a generator f by

$$x \bullet y = f^{-1}\left((f(x) + f(y) - 1) \vee 0\right)$$

The "additive" generator $F : [0,1] \to [0,\infty]$ for nilpotent t-norms that is commonly referred to in the literature is related to f by $F(x)/F(0) = (1 - f(x))$. Thus, there is a decreasing function $F : [0,1] \to [0,\infty]$ with

$$x \bullet y = F^{-1}((F(x) + F(y)) \wedge F(0))$$

A "multiplicative" generator $G : [0,1] \to [0,1]$ is obtained from f by $G(x) = e^{f(x)-1}$, giving a third representation for nilpotent t-norms

$$x \bullet y = G^{-1}((G(x)\,G(y)) \vee G(0))$$

Here are two well-known examples of strict t-norms. For other strict and nilpotent t-norms, see Tables 3.1 (a) and 3.2 (a) starting on page 97.

- Hamacher one-parameter family of t-norms:
  $x \circ_H y = \dfrac{xy}{x + y - xy}$
  generator: $f(x) = e^{\frac{x-1}{x}}$

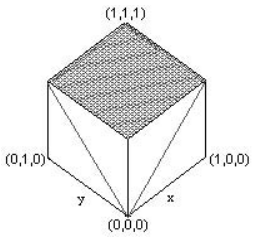
- The Frank one-parameter family of t-norms includes strict, nilpotent, and idempotent t-norms:
  $x \circ_{F_a} y = \log_a \left[1 + \dfrac{(a^x - 1)(a^y - 1)}{a - 1}\right]$, $0 < a < \infty$, $a \neq 1$
  $x \circ_{F_1} y = \lim_{a \to 1} \log_a \left[1 + \frac{(a^x-1)(a^y-1)}{a-1}\right] = xy$
  $x \circ_{F_\infty} y = \lim_{a \to \infty} \log_a \left[1 + \frac{(a^x-1)(a^y-1)}{a-1}\right] = (x + y - 1) \vee 0$
  $x \circ_{F_0} y = \lim_{a \to 0^+} \log_a \left[1 + \frac{(a^x-1)(a^y-1)}{a-1}\right] = x \wedge y$
  generators: $F_a(x) = \dfrac{a^x - 1}{a - 1}$, $0 < a < \infty$, $a \neq 1$; $F_1(x) = x$

Triangular conorms The corresponding generalization for OR is a **triangular conorm** or **t-conorm**.

Definition 3.3 *A **t-conorm** is a binary operation* $* : [0,1] \times [0,1] \to [0,1]$ *satisfying for all* $x, y, z \in [0,1]$

1. $x * y = y * x$ ($*$ *is commutative*)
2. $x * (y * z) = (x * y) * z$ ($*$ *is associative*)
3. $x * 0 = 0 * x = x$ (0 *is an identity*)
4. $y \leq z$ *implies* $x * y \leq x * z$ ($*$ *is increasing in each variable*)

Maximum $x \vee y$ is the smallest (and most widely used) t-conorm. The largest t-conorm, called the **drastic sum**, is not continuous.



Drastic sum

$$x * y = \begin{cases} x \vee y & \text{if} \quad x \wedge y = 0 \\ 1 & \text{if} \quad x \wedge y > 0 \end{cases}$$

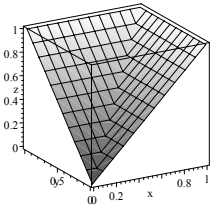
The most common examples, in addition to **maximum**, are the **algebraic sum**

$$x * y = x + y - xy$$

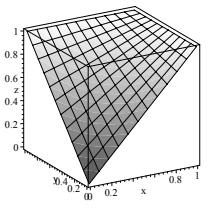
and the **Łukasiewicz t-conorm** or **bounded sum**

$$x * y = (x + y) \wedge 1$$

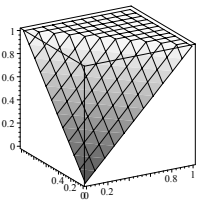
These three t-conorms are depicted in the following plots:



$x \vee y$



$x + y - xy$



$(x + y) \wedge 1$

The three examples depicted above are basic, in the sense that all other continuous t-conorms can be obtained from these three in a straightforward way.

Table 3.1 (a). Strict t-norms, $a > 0$, $r > 0$

Type	**t-norm**	**generator**
Algebraic product	xy	x
Hamacher	$\dfrac{xy}{a+(1-a)(x+y-xy)}$	$\begin{cases} \frac{x}{a-(a-1)x} & a>0 \\ e^{\frac{x-1}{x}} & a=0 \end{cases}$
Frank	$\log_a\left(1+\frac{(a^x-1)(a^y-1)}{a-1}\right)$, $a\neq 1$	$\dfrac{a^x-1}{a-1}$
Schweizer-Sklar	$(x^{-a}+y^{-a}-1)^{-\frac{1}{a}}$	$\exp\left(-\frac{1-x^a}{(2^a-1)x^a}\right)$
Schweizer-Sklar	$1-((1-x)^a+(1-y)^a - (1-x)^a(1-y)^a)^{\frac{1}{a}}$	$1-(1-x)^a$
Aczél-Alsina	$e^{-((-\ln x)^a+(-\ln y)^a)^{\frac{1}{a}}}$	$e^{-r(-\ln x)^a}$, $r>0$
Dombi	$\left(1+\left(\left(\frac{1-x}{x}\right)^a+\left(\frac{1-y}{y}\right)^a\right)^{\frac{1}{a}}\right)^{-1}$	$e^{-\left(\frac{1-x}{x}\right)^a}$
1-parameter family	$xye^{-a\ln x\ln y}$	$\dfrac{1}{1-a\ln x}$
2-parameter family	$\left(1+[(\frac{1-x}{x})^r+(\frac{1-y}{y})^r + a((\frac{1-x}{x})^r(\frac{1-y}{y})^r]^{\frac{1}{r}}\right)^{-1}$	$\left(1+a\left(\frac{1-x}{x}\right)^r\right)^{-1}$

Table 3.1 (b). Strict t-conorms, $a > 0$, $r > 0$

Type	**t-conorm**	**cogenerator**
Algebraic sum	$x+y-xy$	$1-x$
Hamacher	$\dfrac{x+y+(a-2)xy}{1+(a-1)xy}$	$\begin{cases} \frac{1-x}{1+(a-1)x} & a>0 \\ e^{\frac{x}{1-x}} & a=0 \end{cases}$
Frank, $a\neq 1$	$1-\log_a\left(1+\frac{(a^{1-x}-1)(a^{1-y}-1)}{a-1}\right)$	$\frac{a^{1-x}-1}{a-1}$
Schweizer-Sklar	$1-((1-x)^{-a}+(1-y)^{-a}-1)^{-\frac{1}{a}}$	$\exp\left(-\frac{1-(1-x)^a}{(2^a-1)(1-x)^a}\right)$
Schweizer-Sklar	$(x^a+y^a-x^ay^a)^{\frac{1}{a}}$	$1-x^a$
Aczél-Alsina	$e^{-((-\ln x)^a+(-\ln y)^a)^{\frac{1}{a}}}$	$e^{-r(-\ln x)^a}$
Dombi	$\left(1+\left(\left(\frac{1-x}{x}\right)^a+\left(\frac{1-y}{y}\right)^a\right)^{\frac{1}{a}}\right)^{-1}$	$e^{-\left(\frac{1-x}{x}\right)^a}$
1-parameter family	$1-(1-x)(1-y)\,e^{-a\ln(1-x)\ln(1-y)}$	$\dfrac{1}{1-a\ln(1-x)}$
2-parameter family	$\left(1+\left((\frac{1-x}{x})^r+(\frac{1-y}{y})^r + a((\frac{1-x}{x})^r(\frac{1-y}{y})^r\right)^{\frac{1}{r}}\right)^{-1}$	$\left(1+a\left(\frac{1-x}{x}\right)^r\right)^{-1}$

Table 3.2 (a). Nilpotent t-norms, $a > 0$

Type	t-norm	L-generator/Residual
Bounded product	$(x + y - 1) \vee 0$	L-Gen: x Res: $1 - x$
Schweizer-Sklar	$((x^a + y^a - 1) \vee 0)^{\frac{1}{a}}$	L-Gen: x^a Res: $(1 - x^a)^{\frac{1}{a}}$
Yager	$\left(1-((1-x)^a+(1-y)^a)^{\frac{1}{a}}\right) \vee 0$	L-Gen: $1-(1-x)^a$ Res: $1-(1-(1-x)^a)^{\frac{1}{a}}$
Sugeno-Weber	$(a\,(x + y - 1) - (a - 1)\,xy) \vee 0$ $a \neq 1$	L-Gen: $-\log_a\left(1 - x + \frac{1}{a}x\right)$ Res: $\frac{1-x}{1-\frac{a-1}{a}x}$

Table 3.2 (b). Nilpotent t-conorms, $a > 0$

Type	t-conorm	L-cogenerator/Residual
Bounded sum	$(x + y) \wedge 1$	L-Cog: $1 - x$ Res: $1 - x$
Schweizer-Sklar	$1 - (((1 - x)^a +$ $(1 - y)^a - 1) \vee 0)^{\frac{1}{a}}$	L-Cog: $(1 - x)^a$ Res: $1 - (1 - (1 - x)^a)^{\frac{1}{a}}$
Yager	$(x^a + y^a)^{\frac{1}{a}} \wedge 1$	L-Cog: $1 - x^a$ Res: $(1 - x^a)^{\frac{1}{a}}$
Sugeno-Weber	$(x + y + (a - 1)\,xy) \wedge 1$ $a \neq 1$	L-Cog: $1 - \log_a(1 - x + ax)$ Res: $\frac{1-x}{1+(a-1)x}$

A t-conorm that satisfies $x \circ x = x$ for all x is called **idempotent**. The maximum t-conorm is idempotent, and it is the only idempotent t-conorm. A continuous t-conorm that satisfies $x \circ x > x$ for all $x \neq 0, 1$ is called **Archimedean**. All continuous t-conorms we will consider, other than maximum, are Archimedean.

An Archimedean t-conorm for which $x \circ x = 1$ only when $x = 1$ is called **strict**. The algebraic sum t-conorm is the prototype for a strict t-conorm. Archimedean t-conorms that are not strict are **nilpotent**. The bounded sum t-conorm is the prototype for a nilpotent t-conorm. See Tables 3.1 (b) and 3.2 (b) for examples of strict and nilpotent t-conorms.

Nonassociative AND operations Averaging two AND operations produces a new AND operation we will denote by "&." The result is a nonassociative operation that can be used to model real-life expert reasoning.

Example 3.2 Take the Łukasiewicz and max-min t-norms for the two AND operations. If we know the degrees of certainty (subjective probabilities) $p_1 =$

$p(S_1)$ and $p_2 = p(S_2)$ of two statements S_1 and S_2, then possible values of $p_1 \& p_2 = p(S_1 \& S_2)$ form the interval

$$[\max(p_1 + p_2 - 1, 0), \min(p_1, p_2)]$$

As a numerical estimate, we can use a midpoint of this interval

$$p_1 \& p_2 \equiv \tfrac{1}{2} \max(p_1 + p_2 - 1, 0) + \tfrac{1}{2} \min(p_1, p_2)$$

or more generally, we can take a weighted average

$$p_1 \& p_2 \equiv \alpha \max(p_1 + p_2 - 1, 0) + (1 - \alpha) \min(p_1, p_2)$$

where $\alpha \in (0, 1)$. The corresponding OR operations are

$$p_1 (OR) p_2 \equiv \alpha \max(p_1, p_2) + (1 - \alpha) \min(p_1 + p_2, 1)$$

Bouchon-Meunier, Kreinovich and Nguyen argue that these AND operations explain the empirical law in psychology according to which a person can normally distinguish between no more than 7 ± 2 classes of objects. This is related to the fact that in intelligent control, experts normally use ≤ 9 different degrees (such as "small," "medium," etc.) to describe the value of each characteristic (see [10]).

Negations The logical operation $x' = \text{NOT } x$ satisfies $1' = 0$ and $0' = 1$ and $x \leq y$ implies $x' \geq y'$. The operation is a **strong negation** if it also satisfies $(x')' = x$. Such an operation is also called an **involution** or a **duality**. Strong negations are characterized as continuous, strictly decreasing functions $\eta : [0, 1] \to [0, 1]$ that satisfy

$$\begin{aligned} \eta(\eta(x)) &= x \\ \eta(1) &= 0 \\ \eta(0) &= 1 \end{aligned}$$

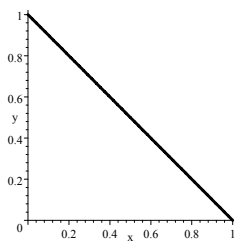
The word negation is often used to mean strong negation. Here are some common examples.

1. Standard: $x' = 1 - x$
2. Sugeno: $x' = \dfrac{1 - x}{1 + ax}$, $a > -1$
3. Yager: $x' = (1 - x^a)^{\frac{1}{a}}$, $a > 0$
4. Exponential: $x' = e^{\frac{a}{\ln x}}$, $a > 0$
5. Logarithmic: $x' = \log_a(a - a^x + 1)$, $a > 0$, $a \neq 1$
6. $x' = 1 - (1 - (1 - x)^a)^{\frac{1}{a}}$, $a > 0$

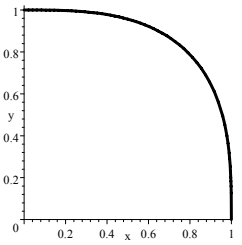
7. $x' = 1 - \log_a\left(a - a^{1-x} + 1\right)$, $a > 0$, $a \neq 1$

8. Gödel: $x' = 0$ if $x \neq 0$, $0' = 1$ (Note $(x')' \neq x$)

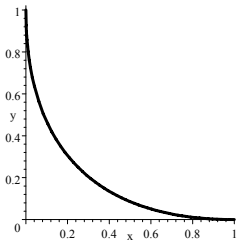
Following are some pictures of strong negations.



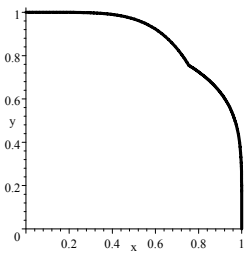
$1 - x$



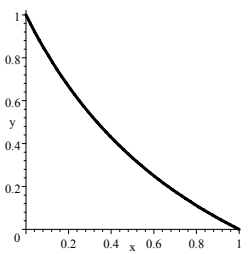
$\left(1 - x^3\right)^{\frac{1}{3}}$



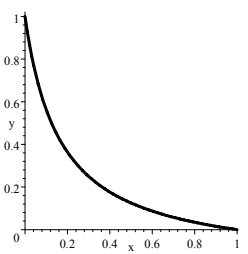
$\left(1 - x^{\frac{1}{2}}\right)^2$



$\begin{cases} 1 - x^5 & \text{if} \quad x \leq 0.755 \\ \sqrt[5]{1-x} & \text{if} \quad x \geq 0.755 \end{cases}$



$\dfrac{1-x}{1+x}$



$\dfrac{1-x}{1+6x}$

Note that these graphs are all symmetric about the line $y = x$. This is a result of the requirement that $\eta(\eta(x)) = x$, which implies that for each point $(x, \eta(x))$ in the graph, the point $(\eta(x), \eta(\eta(x))) = (\eta(x), x)$ is also in the graph. The last two examples are **Sugeno negations** [71], functions of the form

$$\eta(x) = \frac{1-x}{1+\lambda x}, \ \lambda > -1$$

A nilpotent t-norm has a (strong) negation naturally associated with it. This is the "residual" defined by

$$\eta(x) = \bigvee \{y : x \triangle y = 0\}$$

For example, $1 - x$ is the residual of the bounded product.

De Morgan systems If $'$ is a strong negation, the equations

$$\begin{aligned} (x \vee y)' &= x' \wedge y' \\ (x \wedge y)' &= x' \vee y' \end{aligned}$$

both hold. These are called the **De Morgan laws**. They may or may not hold for other pairs of t-norms and t-conorms. When they do, we say that $\circ$ is **dual** to $*$ via the negation $'$, and we call the triple $\circ$, $*$, $'$ a **De Morgan system** on the unit interval. For example, $\wedge$, $\vee$, $1-x$ is a De Morgan system. Tables 3.1 and 3.2 provide strict and nilpotent De Morgan systems if you pair t-norms and t-conorms with common names and parameters. The duality between the t-norm/t-conorm pairs in Tables 3.1 and 3.2 is via the negation $1-x$, except for the Aczél-Alsina case where the duality is via the negation $e^{\frac{1}{\ln x}}$. In the nilpotent case, it is also natural to use the duality given by the t-norm or t-conorm residual, but that is not done here.

The Frank t-norms are the solutions to the functional equation

$$x \triangle y + x \triangledown y = x + y$$

where $x \triangledown y = 1 - ((1-x) \triangle (1-y))$ is the t-conorm dual to $\triangle$ relative to the negation $1-x$. The Frank–De Morgan systems are the triples for which the Frank t-norm and Frank t-conorm use the same parameter and the negation is $1-x$. See page 95 for a complete list of Frank t-norms.

3.3.3 Averaging operators

In the most general sense, averaging operations are aggregation operations that produce a result that lies between the minimum and maximum. Averaging operators represent some kind of compromise, and are of fundamental importance in decision-making. Averaging operators are, in general, not associative.

Definition 3.4 *An **averaging operator** (or **mean**) is a continuous binary operation*

$$\oplus : [0,1] \times [0,1] \to [0,1]$$

satisfying

1. $x \oplus x = x$ (*idempotent*)
2. $x \oplus y = y \oplus x$ (*commutative*)
3. $x \leq y$ *and* $u \leq v$ *implies* $x \oplus u \leq y \oplus v$ (*monotonic*)

For an averaging operator $\oplus$ on $[0,1]$, it is always true that

$$x \wedge y \leq x \oplus y \leq x \vee y$$

In other words, the average of x and y lies *between* x and y. To see this, just observe that

$$x \wedge y = (x \wedge y) \oplus (x \wedge y) \leq x \oplus y \leq (x \vee y) \oplus (x \vee y) = (x \vee y)$$

which follows from the idempotent and monotonic properties of averaging operators. Continuous t-norms and t-conorms are averaging operators. The arithmetic mean $(x+y)/2$ is, of course, an averaging operator, as are geometric

mean $\sqrt{xy}$, harmonic mean $xy/(x+y)$, and median. All averaging operations can be extended to operations on fuzzy sets via

$$(A \oplus B)(x) = A(x) \oplus B(x)$$

The quasi-arithmetic mean is a direct generalization of the arithmetic mean.

Definition 3.5 *A **quasi-arithmetic mean** is a strictly monotonic, continuous averaging operator satisfying*

4. $(x \oplus y) \oplus (z \oplus w) = (x \oplus z) \oplus (y \oplus w)$ ($\dot{+}$ *is* bisymmetric).

Any quasi-arithmetic mean $\oplus$ can be written in the form

$$x \oplus y = f^{-1}\left(\frac{f(x) + f(y)}{2}\right)$$

where f is a continuous, strictly increasing function $f : [0,1] \to [0,1]$ satisfying $f(0) = 0$ and $f(1) = 1$. Technically, every quasi-arithmetic mean is *isomorphic* to the arithmetic mean. Examples of quasi-arithmetic means include the power and logarithmic means:

$$\begin{aligned} x \oplus y &= \left(\frac{x^a + y^a}{2}\right)^{\frac{1}{a}}, \; a > 0 \\ x \oplus y &= \log_a\left(a^x + a^y\right), \; a > 1 \end{aligned}$$

The quasi-arithmetic mean for n points $x_1, x_2, ..., x_n$ is given by

$$f^{-1}\left(\frac{\sum_{i=1}^{n} f(x_i)}{n}\right)$$

Averaging operators suggested by Werners [80] combine the minimum and maximum operators with the arithmetic mean.

Definition 3.6 *The **"fuzzy and"** operator $\hat{\oplus}$ of Werners is given by*

$$x \,\hat{\oplus}\, y = \gamma(x \wedge y) + (1-\gamma)\frac{x+y}{2}$$

*and the **"fuzzy or"** operator $\check{\oplus}$ of Werners is given by*

$$x \,\check{\oplus}\, y = \gamma(x \vee y) + (1-\gamma)\frac{x+y}{2}$$

for some $\gamma \in [0,1]$.

As γ ranges from 0 to 1, the "fuzzy and" ranges from the arithmetic mean to the minimum, and the "fuzzy or" ranges from the arithmetic mean to the maximum.

An operator, suggested by Zimmermann and Zysno [88], that is more general in the sense that the compensation between intersection and union is expressed by a parameter γ, is called "*compensatory and*." This is not an averaging operator in the sense of our definition, since it is not idempotent. However, it plays a similar role in decision making.

Definition 3.7 *The "**compensatory and**" operator is*

$$\left(\prod_{i=1}^{n} x_i\right)^{1-\gamma}\left(1-\prod_{i=1}^{n}(1-x_i)\right)^{\gamma}$$

for some $\gamma \in [0,1]$.

This operator is a combination of the algebraic product and the algebraic sum. The parameter indicates where the actual operator is located between the these two, giving the product when $\gamma = 0$ and the algebraic sum when $\gamma = 1$.

When it is desirable to accommodate variations in the importance of individual items, we can use **weighted generalized means** for the average of $x_1, x_2, ..., x_n$ weighted by the vector $w_1, w_2, ..., w_n$ with $\sum_{i=1}^{n} w_i = 1$, as defined by the formula

$$\left(\sum_{i=1}^{n} w_i x_i^a\right)^{\frac{1}{a}}$$

An aggregation technique, due to Yager [83], uses ordered weighted averaging (OWA) operators, which gives the greatest weight to objects of greatest magnitude.

Definition 3.8 *An **OWA operator of dimension** n, with associated vector $W = (w_1, ..., w_n)$ satisfying $w_i \geq 0$ and $\sum_{j=1}^{n} w_i = 1$, is the mapping*

$$F_W : \mathbb{R}^n \to \mathbb{R} : (x_1, ..., x_n) \mapsto \sum_{j=1}^{n} x_{\sigma(i)} w_i$$

where $\left(a_{\sigma(1)}, ..., a_{\sigma(n)}\right)$ is a rearrangement of the coordinates of $(x_1, ..., x_n)$ so that $x_{\sigma(1)} \geq \cdots \geq x_{\sigma(n)}$.

The rearrangement of the coordinates into an ordered sequence is a crucial part of this definition.

Example 3.3 Assume $W = (0.3, 0.4, 0.3)$. Then,

$$F_W(1,5,3) = (0.3, 0.4, 0.3) \cdot (5,3,1) = 3.0$$

Yager pointed out the following special cases:

- If $W = (1, 0, ..., 0)$, then $F_W(a_1, ..., a_n) = \max\{a_1, ..., a_n\}$.
- If $W = (0, 0, ..., 1)$, then $F_W(a_1, ..., a_n) = \min\{a_1, ..., a_n\}$.
- If $W = \left(\frac{1}{n}, \frac{1}{n}, ..., \frac{1}{n}\right)$, then $F_W(a_1, ..., a_n) = \frac{1}{n}\sum_{j=1}^{n} w_i$.
- If $W = \left(0, \frac{1}{n-2}, ..., \frac{1}{n-2}, 0\right)$, then $F_W(a_1, ..., a_n) =$ the "Olympic average."

3.4 Sensitivity of functions

The modeling of fuzzy concepts through the assignment of membership functions, as well as the choice of fuzzy connectives, is subjective. In specific applications, choices must be made. We illustrate some possible guidelines for making these choices when sensitivity of membership functions or of fuzzy logical connectives with respect to variations in their arguments is a factor. We look at two measures of sensitivity — extreme and average sensitivity.

Note that here, by sensitivity we mean sensitivity with respect to measurement errors or to the flexibility in assigning degrees of membership for fuzzy concepts. Thus, in general, a least sensitive operator among a class of operators is preferred.

3.4.1 Extreme measure of sensitivity

Definition 3.9 *For a mapping $f : [0,1]^n \to [0,1]$ and $\delta \in [0,1]$, let*

$$\rho_f(\delta) = \bigvee_{|x_i - y_i| \le \delta} |f(x) - f(y)|$$

*where $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$. The function $\rho_f : [0,1] \to [0,1]$ is called an **extreme measure of sensitivity** of f. We say that f is **less sensitive** than g if for all δ, $\rho_f(\delta) \le \rho_g(\delta)$, with strict inequality at some δ.*

Example 3.4 If $f(x,y) = x \wedge y$, then for $|x-u| \le \delta$ and $|y-v| \le \delta$, we have $x \le u + \delta$ and $y \le v + \delta$. So

$$(x \wedge y) \le (u+\delta) \wedge (v+\delta) = (u \wedge v) + \delta$$

and thus $(x \wedge y) - (u \wedge v) \le \delta$. Similarly, $(u \wedge v) \le (x \wedge y) + \delta$, so that $(u \wedge v) - (x \wedge y) \le \delta$. Thus, $\rho_\wedge(\delta) \le \delta$.

Taking $x = y = \delta$ and $u = v = 0$, we have

$$(x \wedge y) - (u \wedge v) = (\delta \wedge \delta) - (0 \wedge 0) = \delta$$

so $\rho_\wedge(\delta) \ge \delta$. It follows that

$$\rho_\wedge(\delta) = \delta$$

Here are some additional examples.

Function	Sensitivity
$f(x) = 1 - x$	$\rho_f(\delta) = \delta$
$f(x,y) = xy$	$\rho_f(\delta) = 2\delta - \delta^2$
$f(x,y) = x + y - xy$	$\rho_f(\delta) = 2\delta - \delta^2$
$f(x,y) = (x+y) \wedge 1$	$\rho_f(\delta) = 2\delta \wedge 1$
$f(x,y) = x \vee y$	$\rho_f(\delta) = \delta$

Theorem 3.3 *A t-norm $\triangle$ and the t-conorm $\triangledown$ that is dual to $\triangle$ with respect to $\alpha(x) = 1 - x$, have the same sensitivity.*

Proof. Using the fact that x and $1-x$ have the same range of values for $x \in [0,1]$, we have the equalities

$$\begin{aligned}
\rho_{\triangle}(\delta) &= \bigvee_{|x-u|\vee|y-v|\leq\delta} |x \triangle y - u \triangle v| \\
&= \bigvee_{|(1-x)-(1-u)|\vee|(1-y)-(1-v)|\leq\delta} |(1-x) \triangle (1-y) - (1-u) \triangle (1-v)| \\
&= \bigvee_{|(1-x)-(1-u)|\vee|(1-y)-(1-v)|\leq\delta} |1-(1-x) \triangle (1-y) - (1-(1-u) \triangle (1-v))| \\
&= \bigvee_{|x-u|\vee|y-v|\leq\delta} |x \triangledown y - u \triangledown v| \\
&= \rho_{\triangledown}(\delta)
\end{aligned}$$

Thus $\rho_{\triangle}(\delta) = \rho_{\triangledown}(\delta)$. ■

Theorem 3.4 *The functions $x \wedge y$, $x \vee y$, and $\alpha(x) = 1-x$ are the least sensitive among all continuous Archimedean t-norms and t-conorms, and all negations, respectively.*

Proof. We showed in the first example above that $\rho_{\wedge}(\delta) = \delta$. If $\triangle$ is any t-norm, then

$$\begin{aligned}
|1 \triangle 1 - (1-\delta) \triangle (1-\delta)| &= |1 - (1-\delta) \triangle (1-\delta)| \\
&\leq \rho_{\triangle}(\delta)
\end{aligned}$$

so $(1-\delta) \geq (1-\delta) \triangle (1-\delta) > 1 - \rho_{\triangle}(\delta)$. Thus, $\rho_{\triangle}(\delta) \geq \delta = \rho_{\wedge}(\delta)$.

Note that $\wedge$ is the only t-norm $\triangle$ such that $\rho_{\triangle}(\delta) = \delta$. Indeed, for $\wedge \neq \triangle$, there are x, y such that $x \triangle y \neq x \wedge y$, and we may assume that $x \triangle y < x$. Now

$$|x \triangle 1 - 1 \triangle 1| = 1 - x \triangle y > 1 - x$$

so that $\rho_{\triangle}(1-x) \neq 1-x$. We leave the rest of the proof as exercises. ■

To consider sensitivity of membership functions, let us look at the triangular function A with endpoints $(a,0)$ and $(b,0)$ and high point (c,α) (where $a \leq c \leq b$ and $0 \leq \alpha \leq 1$). This function is defined by

$$A(x) = \begin{cases} \alpha\left(\dfrac{x-a}{c-a}\right) & \text{if } a \leq x \leq c \\ \alpha\left(\dfrac{x-b}{c-b}\right) & \text{if } c \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

Using the fact that if either $x \leq c \leq y$ or $y \leq c \leq x$, and $|x-y| < \delta$, then $|x-c| < \delta$ and $|y-c| < \delta$, we get an upper bound for the sensitivity of this

membership function:

$$\rho_A(\delta) = \bigvee_{|x-y|\le\delta} \left\{ \begin{array}{cc} \left|\alpha\frac{x-y}{c-a}\right| & \text{if } a\le x,y\le c \\ \left|\alpha\frac{x(c-b)-y(c-a)+ac-bc}{(c-a)(c-b)}\right| & \text{if } a\le x\le c\le y\le b \\ \left|\alpha\frac{x(c-a)-y(c-b)+ac-bc}{(c-a)(c-b)}\right| & \text{if } a\le y\le c\le x\le b \\ \left|\alpha\frac{x-y}{c-b}\right| & \text{if } c\le x,y\le b \\ 0 & \text{otherwise} \end{array} \right\}$$

$$\le \left\{ \begin{array}{cc} \frac{\alpha}{c-a}\delta & \text{if } a\le x\le c, a\le y\le c \\ \frac{\alpha(b-a)}{(c-a)(b-c)}\delta & \text{if } a\le x\le c\le y\le b \text{ or } a\le y\le c\le x\le b \\ \frac{\alpha}{b-c}\delta & \text{if } c\le x\le b, c\le y\le b \\ 0 & \text{otherwise} \end{array} \right\}$$

Note that $\frac{\alpha}{c-a}$ and $\frac{\alpha}{b-c}$ are the absolute values of the slopes of the triangular function at the left and right of c, respectively.

For the Gaussian functions $A(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$, the sensitivity function is

$$\rho_A(\delta) = \bigvee_{|x-y|\le\delta} \left| e^{-\frac{(x-c)^2}{2\sigma^2}} - e^{-\frac{(y-c)^2}{2\sigma^2}} \right|$$

and for the sigmoidal functions of the form $f(x) = \frac{1}{1+e^{-(x-m)\sigma}}$, the sensitivity function is

$$\rho_f(\delta) = \bigvee_{|x-y|\le\delta} \left| \frac{1}{1+e^{-(x-m)\sigma}} - \frac{1}{1+e^{-(y-m)\sigma}} \right|$$

3.4.2 Average sensitivity

An alternative to the measure above, of extreme sensitivity of fuzzy logical connectives and membership functions, is a measure of **average sensitivity**. Let $f : [a,b] \to \mathbb{R}$. One measure of the sensitivity of differentiable functions f at a point in $[a,b]$ is the square $f'(x)^2$ of its derivative at that point. Its average sensitivity is the average over all points in $[a,b]$ of $f'(x)^2$, namely, the quantity

$$\frac{1}{b-a}\int_a^b f'(x)^2 dx$$

If f is a function of two variables, say $f : [a,b]^2 \to \mathbb{R}$, the average sensitivity of f is

$$S(f) = \frac{1}{(b-a)^2}\int_a^b\int_a^b \left(\left(\frac{\partial}{\partial x} f(x,y)\right)^2 + \left(\frac{\partial}{\partial y} f(x,y)\right)^2 \right) dxdy$$

Example 3.5 Here are some examples for logical connectives on $[0,1]$.

Connective	Average sensitivity
$\wedge$ *or* $\vee$	$S(\wedge) = S(\vee) = 1$
$x \triangle y = xy$	$S(\triangle) = \frac{2}{3}$
$x \triangle y = x + y - xy$	$S(\triangle) = \frac{2}{3}$
$x \triangle y = (x + y) \wedge 1$	$S(\triangle) = 1$
$x \triangle y = 0 \vee (x + y - 1)$	$S(\triangle) = 1$
$\alpha(x) = 1 - x$	$S(\alpha) = 1$

A t-norm and its dual t-conorm with respect to $\alpha(x) = 1 - x$ have the same average sensitivity. The functions $\wedge$ and $\vee$ in the examples above are differentiable at all points in the unit square except for the line $x = y$, so there is no problem calculating the integrals involved. In certain situations, one may need to use more general notions of derivative.

Theorem 3.5 *The connectives $x \triangle y = xy$, $x \bigtriangledown y = x + y - xy$, and $\alpha(x) = 1 - x$ have the smallest average sensitivity among t-norms, t-conorms, and negations, respectively.*

Proof. We need to show, for example, that $x \triangle y = xy$ minimizes

$$\int_0^1 \int_0^1 \left(\left(\frac{\partial \triangle}{\partial x} \right)^2 + \left(\frac{\partial \triangle}{\partial y} \right)^2 \right) dxdy$$

A standard fact from analysis is that $\triangle$ minimizes this expression if it satisfies the Laplace equation

$$\frac{\partial^2 \triangle}{\partial x^2} + \frac{\partial^2 \triangle}{\partial y^2} = 0$$

and of course it does. Similar arguments apply in the other two cases. ■

As in the case of extreme measure of sensitivity, one can use the notion of average sensitivity as a factor in choosing membership functions for fuzzy concepts. When facing a fuzzy concept such as a linguistic label, one might have a class of possible membership functions suitable for modeling the concept. The membership function within this class that minimizes average sensitivity can be a good choice.

For a membership function $A : [a,b] \to [0,1]$, the average sensitivity is

$$S(A) = \frac{1}{(b-a)} \int_a^b \left(\frac{d}{dx} A(x) \right)^2 dx$$

so the triangular function A with end points $(a, 0)$ and $(b, 0)$ and high point (c, α) has average sensitivity

$$\begin{aligned} S(A) &= \frac{1}{(b-a)}\left(\int_a^c \left(\frac{\alpha}{c-a}\right)^2 dx + \int_c^b \left(\frac{\alpha}{c-b}\right)^2 dx\right) \\ &= \frac{1}{(b-a)}\left(\frac{\alpha^2}{c-a} + \frac{\alpha^2}{b-c}\right) \\ &= \frac{\alpha}{(c-a)}\frac{\alpha}{(b-c)} \end{aligned}$$

which is the product of the absolute values of the slopes of the sides of the triangle.

For the sigmoidal membership function $f(x) = \frac{1}{1+e^{-x+1}}$ on the interval $[-5, 10]$, the average sensitivity is

$$\begin{aligned} S(f) &= \frac{1}{(10-(-5))}\int_{-5}^{10}\left(\frac{d}{dx}\frac{1}{1+e^{-x+1}}\right)^2 dx \\ &= \frac{1}{15}\int_{-5}^{10}\left(\frac{e^{-x+1}}{(1+e^{-x+1})^2}\right)^2 dx \\ &= 0.233\,33 \end{aligned}$$

and for the Gaussian membership function $f(x) = e^{-\frac{x^2}{2}}$ on the interval $[-5, 5]$, the average sensitivity is

$$\begin{aligned} S(f) &= \frac{1}{(5-(-5))}\int_{-5}^{5}\left(\frac{d}{dx}e^{-\frac{x^2}{2}}\right)^2 dx \\ &= \frac{1}{10}\int_{-5}^{5}\left(-xe^{-\frac{1}{2}x^2}\right)^2 dx \\ &= 8.862\,3 \times 10^{-2} \end{aligned}$$

3.5 Combining fuzzy rules

The rules used in a rule-based system are generally expressed in a form such as "If x is A then y is B," where A and B are fuzzy sets, x is in the domain of A, and y is in the domain of B. This sounds like an implication, such as "A implies B." There are many generalizations of the classical logical implication operation to fuzzy sets, but most inference mechanisms used in fuzzy logic control systems are not, strictly speaking, generalizations of classical implication.

The reasoning applied in fuzzy logic is often described in terms of a **generalized modus ponens**

Premise 1	x is $\tilde{A}$
Premise 2	If x is A then y is B
Conclusion	y is $\tilde{B}$

where $A, \tilde{A}, B, \tilde{B}$ are fuzzy sets representing fuzzy concepts. The computation of $\tilde{B}$ can be carried out through a basic rule of inference called the **compositional rule of inference**, namely, $\tilde{B} = R \circ \tilde{A}$ where R is a fuzzy relation representing the implication or fuzzy conditional proposition "Premise 2." This inference scheme is sometimes described as a problem of **interpolation**. Interpolation lies at the heart of the utility of fuzzy rule-based systems because it makes it possible to employ a relatively small number of fuzzy rules to characterize a complex relationship between two or more variables.

A number of formulas have been proposed for this implication, most commonly the **compositional conjunction**

$$R(x, y) = A(u) \wedge B(v)$$

Then $\tilde{B}$ is defined as

$$\tilde{B}(v) = \left(R \circ \tilde{A}\right)(u) = \bigvee_u \left(\tilde{A}(u) \wedge A(u) \wedge B(v)\right)$$

(See page 118 for a discussion of max-min composition with fuzzy relations.)

We describe four inference and aggregation mechanisms — named after Mamdani, Larsen, Takagi-Sugeno-Kang, and Tsukamoto — commonly used to interpret a collection of rules

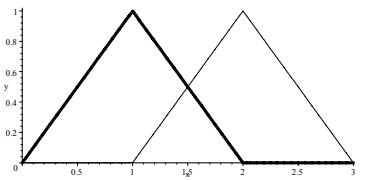
$$\text{If } x \text{ is } A_i \text{ then } y \text{ is } B_i,\ i = 1, 2, ..., n$$

Applications of these methods in control theory will be discussed in Chapter 4.

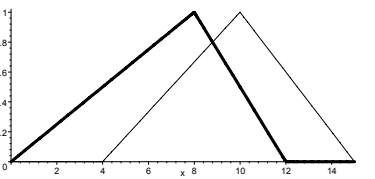
In the following examples, we will use the same four fuzzy sets A_1, A_2, B_1, and B_2 to illustrate the combination of fuzzy rules:

$$A_1(x) = \begin{cases} x & \text{if } 0 \le x \le 1 \\ 2 - x & \text{if } 1 \le x \le 2 \\ 0 & \text{otherwise} \end{cases} \qquad B_1(y) = \begin{cases} \frac{1}{8}y & \text{if } 0 \le y \le 8 \\ -\frac{1}{4}y + 3 & \text{if } 8 \le y \le 12 \\ 0 & \text{otherwise} \end{cases}$$

$$A_2(x) = \begin{cases} x - 1 & \text{if } 1 \le x \le 2 \\ 3 - x & \text{if } 2 \le x \le 3 \\ 0 & \text{otherwise} \end{cases} \qquad B_2(y) = \begin{cases} \frac{1}{6}y - \frac{2}{3} & \text{if } 4 \le y \le 10 \\ -\frac{1}{5}y + 3 & \text{if } 10 \le y \le 15 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$



A_1 and A_2



B_1 and B_2

First we look at the *product* of fuzzy sets, as this greatly simplifies the presentation of fuzzy rules for the Mamdani and Larsen models.

3.5.1 Products of fuzzy sets

The (Cartesian) **product of ordinary sets** $X_1, X_2, ..., X_n$ is the set of n-tuples

$$X_1 \times X_2 \times \cdots \times X_n = \prod_{i=1}^{n} X_i = \{(x_1, x_2, ..., x_n) \mid x_i \in X_i\}$$

The **product of fuzzy sets** $A_i : X_i \to [0,1]$, $i = 1, ..., n$, is the fuzzy set

$$A : \prod_{i=1}^{n} X_i \to [0,1]$$

defined by

$$A(x_1, x_2, ..., x_n) = A_1(x_1) \wedge \cdots \wedge A_n(x_n)$$

Given rules

$$R_i : \text{If } A_{i1} \text{ and } A_{i2} \text{ and ... and } A_{ik} \text{ then } B_i,\ i = 1, 2, ..., n$$

where $A_{ij} : X_j \to [0,1]$ and $B_i : Y \to [0,1]$, interpreting "and" as minimum, we can represent these rules as

$$R_i : \text{If } A_i \text{ then } B_i,\ i = 1, 2, ..., n$$

where $A_i = \prod_{j=1}^{n} A_{ij} : \prod_{j=1}^{n} X_j \to [0,1]$; so in a situation like this, we can always assume that we have just one fuzzy set A_i for each i, with domain $X = \prod_{j=1}^{k} X_j$ an ordinary product of sets. All of the models we describe allow this simplification. A rule R_i is said to **fire** at x if $A_i(x) \neq 0$, in other words, if x is in the support of A_i.

3.5.2 Mamdani model

Given rules "If $\mathbf{x}$ is A_i then y is B_i," $i = 1, ..., n$ where $\mathbf{x} = (x_1, x_2, ..., x_k)$, they are combined in the Mamdani model as

$$R(\mathbf{x}, y) = \bigvee_{i=1}^{n} (A_i(\mathbf{x}) \wedge B_i(y))$$

For each k-tuple $\mathbf{x} = (x_1, x_2, ..., x_k)$ this gives a fuzzy set $R_{\mathbf{x}}$ defined by

$$R_{\mathbf{x}}(y) = \bigvee_{i=1}^{n} A_i(\mathbf{x}) \wedge B_i(y)$$

Note that for the expanded set of rules

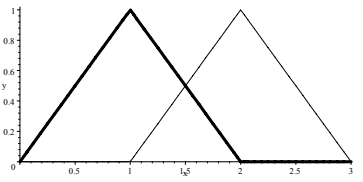
$$R_i : \text{If } A_{i1} \text{ and } A_{i2} \text{ and ... and } A_{ik} \text{ then } B_i,\ i = 1, 2, ..., n$$

this looks like

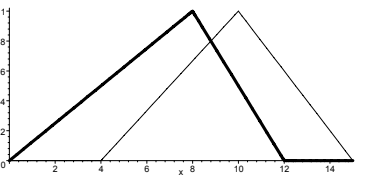
$$R_{\mathbf{x}}(y) = R(x_1, x_2, ..., x_k, y) = \bigvee_{i=1}^{n} (A_{i1}(x_1) \wedge A_{i2}(x_2) \wedge \cdots \wedge A_{ik}(x_k) \wedge B_i(y))$$

In fuzzy control, the number $\bigvee_{i=1}^{n} A_i(\mathbf{x}) = A_{i1}(x_1) \wedge A_{i2}(x_2) \wedge \cdots \wedge A_{ik}(x_k)$ is called the **strength** of the rule R_i for the input $\mathbf{x}$. The fuzzy set $R_{i,\mathbf{x}}(y) = A_i(\mathbf{x}) \wedge B_i(y)$ is called the **control output** of the rule R_i for the input $\mathbf{x}$, and the fuzzy set $R_{\mathbf{x}}(y)$ is the **aggregated control output** for the input $\mathbf{x}$.

Example 3.6 Take the fuzzy sets A_i and B_i defined in Equation 3.8.



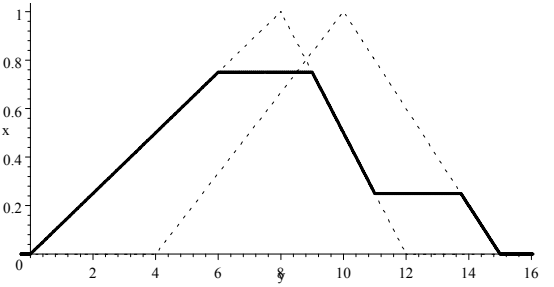
A_1 and A_2



B_1 and B_2

At the point $x = 1.25$, the rules "If x is A_i then y is B_i, $i = 1, 2$," produce the fuzzy set

$$R_{1.25}(y) = \begin{cases} \left(\frac{3}{4} \wedge \frac{1}{8}y\right) & \text{if} \quad 0 \le y \le 4 \\ \left(\frac{3}{4} \wedge \left(\frac{1}{8}y\right)\right) \vee \left(\frac{1}{4} \wedge \left(\frac{1}{6}y - \frac{2}{3}\right)\right) & \text{if} \quad 4 \le y \le 8 \\ \left(\frac{3}{4} \wedge \left(-\frac{1}{4}y + 3\right)\right) \vee \left(\frac{1}{4} \wedge \left(\frac{1}{6}y - \frac{2}{3}\right)\right) & \text{if} \quad 8 \le y \le 10 \\ \left(\frac{3}{4} \wedge \left(-\frac{1}{4}y + 3\right)\right) \vee \left(\frac{1}{4} \wedge \left(-\frac{1}{5}y + 3\right)\right) & \text{if} \quad 10 \le y \le 12 \\ \left(\frac{1}{4} \wedge \left(-\frac{1}{5}y + 3\right)\right) & \text{if} \quad 12 \le y \le 15 \\ 0 & \text{if} \quad \text{otherwise} \end{cases}$$



$[A_1(1.25) \wedge B_1(y)] \vee [A_2(1.25) \wedge B_2(y)]$
B_1 and B_2 (dotted lines)

3.5.3 Larsen model

Given rules "If x is A_i then y is B_i," $i = 1, ..., n$, they are combined in the Larsen model as

$$R(\mathbf{x}, y) = \bigvee_{i=1}^{n} (A_i(\mathbf{x}) \cdot B_i(y))$$

where $\cdot$ indicates multiplication. For each k-tuple $\mathbf{x} = (x_1, x_2, ..., x_k)$ this gives a fuzzy set

$$R_{\mathbf{x}}(y) = \bigvee_{i=1}^{n} A_i(\mathbf{x}) \cdot B_i(y)$$

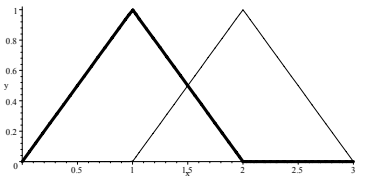
Note that for the set of rules

$$R_i : \text{If } A_{i1} \text{ and } A_{i2} \text{ and ... and } A_{ik} \text{ then } B_i,\ i = 1, 2, ..., n$$

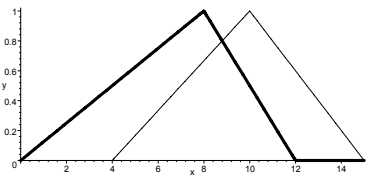
this looks like

$$R_{\mathbf{x}}(y) = R(x_1, x_2, ..., x_k, y) = \bigvee_{i=1}^{n} (A_{i1}(x_1) \wedge A_{i2}(x_2) \wedge \cdots \wedge A_{ik}(x_k)) \cdot B_i(y)$$

Example 3.7 Take the fuzzy sets A_i and B_i defined in Equation 3.8.

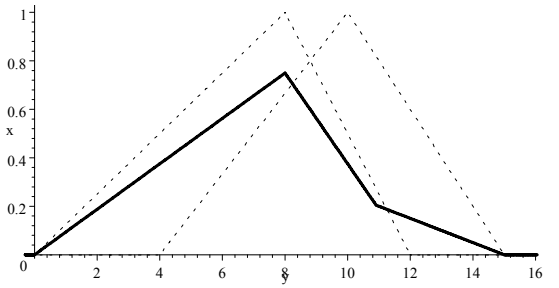


A_1 and A_2



B_1 and B_2

At the point $x = 1.25$, the rules "If x is A_i then y is B_i," $i = 1, 2$, produce the fuzzy set



$R_{1.25}(y) = (A_1(1.25) \cdot B_1(y)) \vee (A_2(1.25) \cdot B_2(y))$
B_1 and B_2 (dotted lines)

3.5.4 Takagi-Sugeno-Kang (TSK) model

For the TSK model, rules are given in the form

$$R_i : \text{If } x_1 \text{ is } A_{i1} \text{ and } x_2 \text{ is } A_{i2} \text{ and ... and } x_k \text{ is } A_{ik}$$
$$\text{then } f_i(x_1, x_2, ..., x_k),\ i = 1, 2, ..., n$$

or

$$R_i : \text{If } \mathbf{x}_i \text{ is } A_i \text{ then } f_i(\mathbf{x}),\ i = 1, 2, ..., n$$

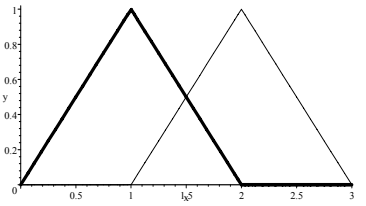
where $f_1, f_2, ..., f_n$ are functions $X = X_1 \times X_2 \times \cdots \times X_k \to \mathbb{R}$ and $A_i = \bigwedge_{j=1}^{k} A_{ij}$. These rules are combined to get a function

$$R(\mathbf{x}) = \frac{A_1(\mathbf{x}) f_1(\mathbf{x}) + A_2(\mathbf{x}) f_2(\mathbf{x}) + \cdots + A_n(\mathbf{x}) f_n(\mathbf{x})}{A_1(\mathbf{x}) + A_2(\mathbf{x}) + \cdots + A_n(\mathbf{x})}$$

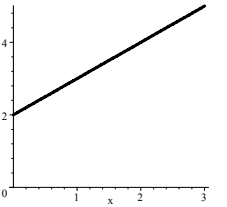
Thus, this model produces a real-valued function.

Example 3.8 Take the fuzzy sets A_i defined in Equation 3.8 and the functions

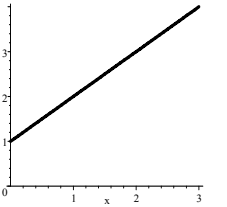
$$f_1(x) = 2 + x \qquad f_2(x) = 1 + x$$



A_1 and A_2



$f_1(x) = 2 + x$

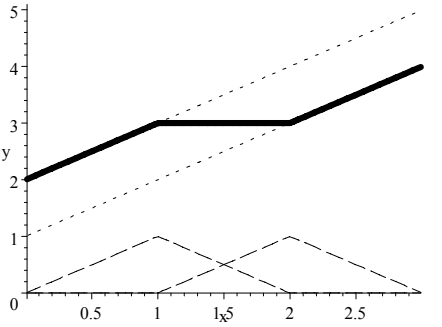


$f_2(x) = 1 + x$

Then the rules

$$R_i : \text{If } x_i \text{ is } A_i \text{ then } f_i(x),\ i = 1, 2$$

produce the function



$$R(x) = \frac{A_1(x) f_1(x) + A_2(x) f_2(x)}{A_1(x) + A_2(x)}$$

A_1 and A_2 (dashed lines)
f_1 and f_2 (dotted lines)

3.5.5 Tsukamoto model

Given rules "If x is A_i then y is C_i," $i = 1, ..., n$, with C_i all monotonic (either strictly increasing or strictly decreasing), the Tsukamoto model produces the function

$$y = \frac{\sum_{i=1}^{n} C_i^{-1} A_i(x)}{\sum_{i=1}^{n} A_i(x)}$$

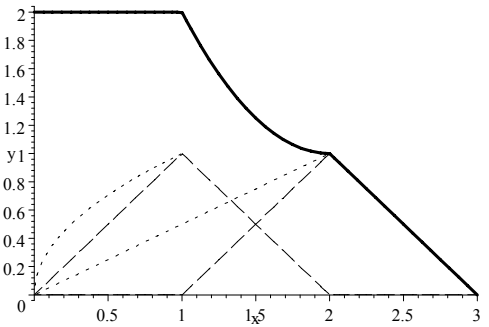
The monotonicity of the fuzzy sets C_i is necessary in order to compute the inverse functions C_i^{-1}.

Example 3.9 Take the two fuzzy sets A_1 and A_2 of Equation 3.8, and the fuzzy sets C_1 and C_2 and their inverses.

$$\begin{array}{ll} C_1(y) = y/2 & C_1^{-1}(z) = 2z \\ C_2(y) = \sqrt{y} & C_2^{-1}(z) = z^2 \end{array}$$

The rules "If x is A_i then y is C_i," $i = 1, 2$, produce the function

$$R(x) = \frac{\sum_{i=1}^{2} C_i^{-1}(A_i(x))}{\sum_{i=1}^{2} A_i(x)} = \begin{cases} 2 & \text{if } 0 \le x \le 1 \\ 5 - 4x + x^2 & \text{if } 1 \le x \le 2 \\ 3 - x & \text{if } 2 \le x \le 3 \end{cases}$$



$$R(x) = \frac{\sum_{i=1}^{2} C_i^{-1}(A_i(x))}{\sum_{i=1}^{2} A_i(x)}$$
A_1 and A_2 (dashed lines)
C_1 and C_2 (dotted lines)

3.6 Truth tables for fuzzy logic

In classical two-valued logic, truth tables can be used to distinguish between expressions. The following tables define the operations of $\vee$, $\wedge$, and $'$ on the truth values $\{0, 1\}$, where 0 is interpreted as "false" and 1 is interpreted as "true."

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

	$'$
0	1
1	0

Truth tables, in a slightly different form, can be used to determine equivalence of logical expressions. Take two variables x, y in classical two-valued logic, let $p = x$ and $q = (x \wedge y) \vee (x \wedge y')$, and compare their values:

x	y	$x \wedge y$	y'	$x \wedge y'$	q	p
0	0	0	1	0	0	0
0	1	0	0	0	0	0
1	0	0	1	1	1	1
1	1	1	0	0	1	1

The fact that q and p have the same truth values for every truth value of x and y reflects the well-known fact that p and q are logically equivalent.

For fuzzy logic, it is a remarkable fact that *three-valued* truth tables can be used for this purpose. Take truth values $\{0, u, 1\}$, where 0 is interpreted as "false," 1 is interpreted as "true," and u as "other." For a fuzzy set, the truth value u corresponds to the case where the degree of membership, or the truth value, is greater than 0 but less than 1. The corresponding truth tables are

$\vee$	0	u	1
0	0	u	1
u	u	u	1
1	1	1	1

$\wedge$	0	u	1
0	0	0	0
u	0	u	u
1	0	u	1

	$'$
0	1
u	u
1	0

These three-valued truth tables can also be used to determine equivalence of logical expressions of fuzzy sets.

Example 3.10 Let $A, B : X \to [0,1]$ be fuzzy sets and consider the expressions

$$\begin{aligned} p(x) &= A(x) \wedge A'(x) \\ q(x) &= (A(x) \wedge A'(x) \wedge B(x)) \vee (A(x) \wedge A'(x) \wedge B'(x)) \end{aligned}$$

where $A'(x) = 1 - A(x)$ and $B'(x) = 1 - B(x)$. Also let

$$\begin{aligned} r(x) &= A(x) \wedge A'(x) \wedge B(x) \\ s(x) &= A(x) \wedge A'(x) \wedge B'(x) \end{aligned}$$

so that $q(x) = r(x) \vee s(x)$. To build a three-valued truth table for a fuzzy set $A : X \to [0,1]$, enter 1 if $A(x) = 1$, 0 if $A(x) = 0$, and u otherwise. Likewise with $B : X \to [0,1]$. Then take all possible values in $\{0, u, 1\}$ for $A(x)$ and $B(x)$ in the first two columns and evaluate the other expressions at those values.

$A(x)$	$B(x)$	$A'(x)$	$B'(x)$	$p(x)$	$r(x)$	$s(x)$	$q(x)$
0	0	1	1	0	0	0	0
u	0	u	1	u	0	u	u
1	0	0	1	0	0	0	0
0	u	1	u	0	0	0	0
u	u	u	u	u	u	u	u
1	u	0	u	0	0	0	0
0	1	1	0	0	0	0	0
u	1	u	0	u	u	0	u
1	1	0	0	0	0	0	0

Observing that the truth values for the expressions $p(x)$ and $q(x)$ are identical, we can conclude that

$$A(x) \wedge A'(x) = (A(x) \wedge A'(x) \wedge B(x)) \vee (A(x) \wedge A'(x) \wedge B'(x))$$

for all $x \in [0,1]$, or in terms of the fuzzy sets,

$$A \wedge A' = (A \wedge A' \wedge B) \vee (A \wedge A' \wedge B')$$

You will be asked to verify this identity for specific fuzzy sets in the exercises. A proof of the general result is contained in *[26].*

3.7 Fuzzy partitions

There is no standardized interpretation of the term "fuzzy partition." In practice, the term is often used to mean any collection $\{A_i : X_i \to [0,1]\}$ of fuzzy sets for which $\cup_i X_i$ is the desired universe. It may also be assumed that for each i, $A_i(x) = 1$ for some x, that is, A_i is **normal**. Sometimes the term is used in one of several ways that more accurately generalize the notion of partition of ordinary sets. We describe some of these other notions after reviewing the classical definition of partition.

For ordinary sets, the term "partition" has a universally accepted meaning. A partition of an ordinary set is a division of the set into nonoverlapping nonempty pieces. There are many occasions, for example, when we divide a set of people into nonoverlapping sets, such as age groups or gender groups. Here is the formal definition.

Definition 3.10 *A finite set $\{A_1, A_2, ..., A_n\}$ of nonempty subsets of a set X is a **partition** of X if the following two conditions are satisfied:*

1. $A_1 \cup A_2 \cup \cdots \cup A_n = X$
2. $A_i \cap A_j = \emptyset$ *if* $i \neq j$

If we regard A_i as a characteristic function, this is equivalent to the two conditions

1. $\sum_{i=1}^{n} A_i(x) = 1$
2. $A_i(x)A_j(x) = 0$ (or equivalently, $A_i(x) \wedge A_j(x) = 0$) for all $x \in X$, $i \neq j$

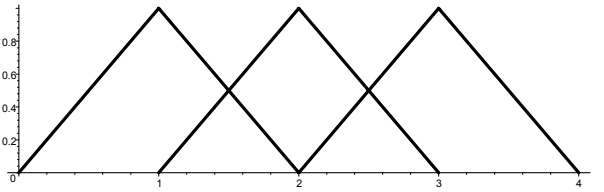
In extending the notion of partition to fuzzy sets, we cannot simply use properties 1 and 2 since having the two conditions (1) $\max\{A_i(x) : i = 1, ..., n\} = 1$ and (2) $\min\{A_i(x), A_j(x)\} = 0$ for $i \neq j$, would imply that for all i and x, $A_i(x) = 0$ or 1 and thus that the A_i are crisp sets. Call a fuzzy set A **normal** if $A(x) = 1$ for some x. This condition together with $\sum_{i=1}^{n} A_i(x) = 1$ does lead to a good definition of partition for fuzzy sets.

Definition 3.11 *A finite set of normal fuzzy subsets* $\{A_1, A_2, ..., A_n\}$ *of* U *is a* ***finite fuzzy partition*** *of a set* U *if*

1. $\sum_{i=1}^{n} A_i(x) = 1$ *for all* $x \in U$

2. *Each* A_i *is normal; that is, for each* i, $A_i(x_i) = 1$ *for some* x_i

This definition captures the meaning of properties 1 and 2 above in the following sense. Each x has a nonzero membership value for some A_i. Also, if $A_i(x) = 1$ for some i, then it is 0 for all others. A typical picture of a fuzzy partition is as follows.



Here, the set $X = [0, 4]$ is partitioned into three fuzzy (triangular) subsets.

For a finite set $X = \{x_1, ..., x_m\}$, with $m \geq n \geq 2$, condition 2 is sometimes replaced by condition 2′:

2′. For each i, $0 < \sum_{k=1}^{m} A_i(x_k) < m$.

See [86] for example. This is a weaker condition. If 1 and 2 are satisfied, take $j \neq i$. There is an x_s with $A_j(x_s) = 1$, and hence $A_i(x_s) = 0$, so that $\sum_{k=1}^{m} A_i(x_k) < m$. Also there is an x_t for which $A_i(x_t) = 1$, so that $\sum_{k=1}^{m} A_i(x_k) > 0$. Thus 2′ is satisfied. However, this weaker condition does not force the A_i to be normal, and does not force each x to have a nonzero membership value for any A_i.

3.8 Fuzzy relations

A *relation* is a mathematical description of a situation where certain elements of sets are related to one another in some way. There are many special kinds of relations — products, functions, equivalence relations, partial orders — to name a few.

Definition 3.12 *Let* $X_1, \ldots, X_n$ *be ordinary sets. An* ***n-ary relation in*** $X_1 \times X_2 \times \cdots \times X_n$ *is a subset* $R \subseteq X_1 \times X_2 \times \cdots \times X_n$. *If* $X_1 = \cdots = X_n = X$, *a subset* $R \subseteq X \times X \times \cdots \times X$ *is an* ***n-ary relation on*** X. *A 2-ary relation is called a* ***binary relation****.*

A ***fuzzy n-ary relation in*** $X_1 \times X_2 \times \cdots \times X_n$ *is a fuzzy subset* $R : X_1 \times X_2 \times \cdots \times X_n \to [0, 1]$, *and a* ***fuzzy n-ary relation on*** X *is a fuzzy subset* $R : X \times X \times \cdots \times X \to [0, 1]$.

Example 3.11 If $X = \{2, 3, 4, 6, 8\}$ and R is the relation "$(x, y) \in R$ if and only if x divides y," then

$$R = \{(2,2), (2,4), (2,6), (2,8), (3,3), (3,6), (4,4), (4,8), (6,6), (8,8)\}$$

Example 3.12 If X is the set of real numbers and Q is the fuzzy binary relation on X described by "$Q(x, y)$ is the degree to which x is close to y," then one possibility for Q is the function

$$Q(x, y) = \frac{1}{|x - y| + 1}$$

so that $Q(x, x) = 1$ for all x, while $Q(2, 8) = \frac{1}{7}$.

If X is finite, say $X = \{x_1, x_2, \ldots, x_n\}$, a fuzzy relation can be represented as a matrix with ij entry $R(x_i, x_j)$. This matrix will have entries all 0s and 1s if and only if the relation is an ordinary relation. For example, the relation R above on $X = \{2, 3, 4, 6, 8\}$ would be represented as

$$\begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 4 & 6 & 8 \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{array}\right] \end{array}$$

while the relation Q, restricted to the same domain, would be represented as

$$\begin{array}{c} \\ 2 \\ 3 \\ 4 \\ 6 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccccc} 2 & 3 & 4 & 6 & 8 \end{array} \\ \left[\begin{array}{ccccc} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{4} & \frac{1}{3} & \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{5} & \frac{1}{4} & \frac{1}{3} & \frac{1}{2} & 1 \end{array}\right] \end{array}$$

Since fuzzy relations are special kinds of fuzzy sets, all of the methods of combining fuzzy sets can be applied to fuzzy relations. In addition, however, there is a composition for fuzzy relations.

Definition 3.13 *If R is a fuzzy binary relation in $X \times Y$ and Q is a fuzzy binary relation in $Y \times Z$, the* ***sup-min composition*** *or* ***max-min composition*** *of R and Q is defined as*

$$(R \circ Q)(x, z) = \bigvee_{y \in Y} (R(x, y) \wedge Q(y, z))$$

Example 3.13 Let $X = \{x_1, x_2, x_3\}$,

$$R = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} 0.9 & 0.2 & 0.2 \\ 0.9 & 0.4 & 0.5 \\ 1.0 & 0.6 & 1.0 \end{bmatrix} \end{array}, \; Q = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.8 & 0 \\ 0 & 0.6 & 1.0 \\ 0.3 & 0.8 & 0.2 \end{bmatrix} \end{array}$$

then $R \circ Q$ has ij entry $\bigvee_{k=1}^{3} (R(x_i, x_k) \wedge Q(x_k, x_j))$ so that

$$R \circ Q = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.8 & 0.2 \\ 0.3 & 0.8 & 0.4 \\ 0.3 & 0.8 & 0.6 \end{bmatrix} \end{array}$$

Notice the similarity to matrix product. Use essentially the same algorithm, but replace product by minimum and sum by maximum.

When $S \subseteq X \times Y$ is a relation, there are two natural maps called projections:

$$\begin{aligned} \pi_X &: S \to X : (x, y) \mapsto x \\ \pi_Y &: S \to Y : (x, y) \mapsto y \end{aligned}$$

In the case of a fuzzy relation $R : X \times Y \to [0, 1]$, the **projections** of R on X and Y are the fuzzy subsets $\pi_X(R)$ of X and $\pi_Y(R)$ of Y defined by

$$\begin{aligned} \pi_X(R)(x) &= \bigvee \{R(x, y) \mid y \in Y\} \\ \pi_Y(R)(y) &= \bigvee \{R(x, y) \mid x \in X\} \end{aligned}$$

3.8.1 Equivalence relations

The fundamental similarity relation is an *equivalence* relation. With an ordinary set, an equivalence relation *partitions* the set into separate pieces, any two members of one of these pieces being *equivalent*. The formal definition is the following.

Definition 3.14 *An **equivalence relation** is a subset $S \subseteq X \times X$, such that for all $x, y, z \in X$*

1. *$(x, x) \in S$ (S is reflexive.)*
2. *$(x, y) \in S$ if and only if $(y, x) \in S$ (S is symmetric.)*
3. *$(x, y) \in S$ and $(y, z) \in S$ implies $(x, z) \in S$ (S is transitive.)*

This has been generalized to fuzzy sets as follows.

Definition 3.15 *A **fuzzy equivalence relation** is a fuzzy relation S on a set X, such that for all $x, y, z \in X$*

1. *$S(x, x) = 1$ (S is reflexive.)*
2. *$S(x, y) = S(y, x)$ (S is symmetric.)*
3. *$S(x, z) \geq S(x, y) \wedge S(y, z)$ (S is max-min transitive.)*

3.8.2 Order relations

A partial order $\leq$ on a set determines a relation R by $(x, y) \in R$ if and only if $x \leq y$. There are many kinds of orders, and many generalizations have been proposed for fuzzy sets. We will mention only the following.

Definition 3.16 *A **fuzzy order relation** is a fuzzy relation S on a set X such that for all $x, y, z \in X$*

1. $S(x, x) = 1$ *(S is reflexive.)*
2. *If $x \neq y$, then $S(x, y) \wedge S(y, x) = 0$ (S is antisymmetric.)*
3. $S(x, z) \geq S(x, y) \wedge S(y, z)$ *(S is max-min transitive.)*

   A fuzzy order relation is a fuzzy linear ordering if it also satisfies

4. *If $x \neq y$, then $S(x, y) \vee S(y, x) > 0$*

3.9 Defuzzification

The most common methods for combining fuzzy rules produce a fuzzy set. In control theory, a crisp output is often needed. This requires some process of **defuzzification** — producing a number that best reflects the fuzzy set in some sense. There are many techniques for defuzzification. We will mention only a few of the most common ones here. We will demonstrate each of these on the output set obtained by the Mamdani method in Example 3.6.

Loosely speaking, there are two types of defuzzification techniques — composite moments and composite maximum. "Composite" reflects the fact that the values are obtained from combining several fuzzy sets. Composite moment techniques use some aspect of the first moment of inertia, and composite maximum techniques extract a value for which the fuzzy set attains its maximum. The center of area and height-center of area methods are of the first type, and the max criterion, first of maxima, and middle of maxima methods are of the second type.

3.9.1 Center of area method

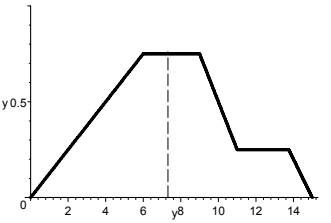
The **center of area**, or **center of gravity**, or **centroid** method computes the center of area of the region under the curve defined by a fuzzy set and selects the first component. If C is the fuzzy set in question and C is integrable, then the defuzzified value of C by this method is

$$z_0 = \frac{\int_a^b zC(z)\,dz}{\int_a^b C(z)\,dz}$$

where $[a, b]$ is an interval containing the support of C. If the support of C is finite, the computation is

$$z_0 = \frac{\sum_{j=1}^{n} z_j C(z_j)}{\sum_{j=1}^{n} C(z_j)}$$

The output set obtained in Example 3.6 produces the following value.



$$z_0(1.25) = 7.3187$$

The center of area defuzzification is the most widely used technique. The defuzzified values tend to move smoothly in reaction to small changes, and it is relatively easy to calculate.

This solution is reminiscent of statistical decision theory. If we normalize $C(\cdot|x) = C^x$, we obtain the probability density function

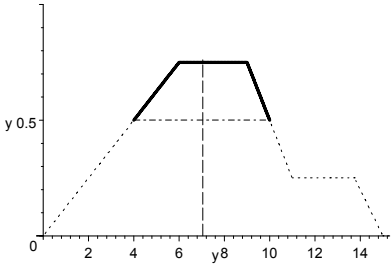
$$\frac{C(\cdot|x)}{\int_Z C(z|x)\,dz}$$

A common way to summarize a distribution is to use its center of location — that is, its expected value:

$$z_0 = \frac{\int_Z zC(z|x)\,dz}{\int_Z C(z|x)\,dz}$$

3.9.2 Height-center of area method

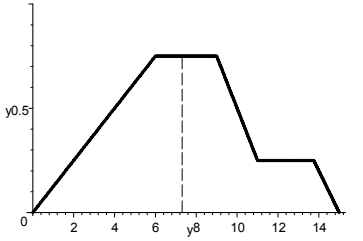
The height-center of area defuzzification method ignores values of the fuzzy set below some level α, then uses the center of area method on the resulting curve. For $\alpha = 0.5$, the output set obtained in Example 3.6 produces the value $z_0(1.25) = 7.5$.



$z_0\,(1.25) = 7.\,060\,6$, height above $\alpha = 0.5$

3.9.3 Max criterion method

This method chooses an arbitrary value from the set of values in the domain on which the fuzzy set assumes its maximum. The output set obtained in Example 3.6 produces a value $6 \leq z_0\,(1.25) \leq 9$.

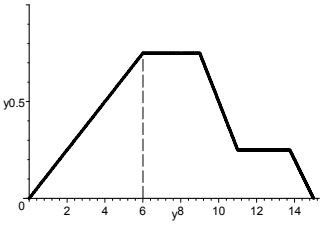


$z_0\,(1.25) \in [6, 9]$

A fuzzy set could be considered as a "possibility distribution" of the variable z. From common sense, we could take z_0 to be a value at which the degree of possibility is the highest. This is the max-criterion defuzzification method.

3.9.4 First of maxima method

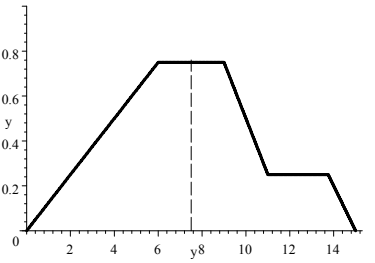
First of maxima takes the smallest value in the domain on which the fuzzy set assumes its maximum. The output set obtained in Example 3.6 produces the value $z_0\,(1.25) = 6$.



$$z_0\,(1.25) = 6$$

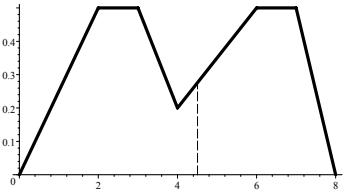
3.9.5 Middle of maxima method

Middle of maxima, or mean of maxima, takes the average of the smallest and largest values in the domain on which the fuzzy set assumes its maximum. The output set obtained in Example 3.6 produces the value $z_0\,(1.25) = \frac{6+9}{2} = 7.5$.



$$z_0\,(1.25) = 7.5$$

This method appears weak, however, for a fuzzy set with two separate plateaus at the maximum height.



$$z_0\,(1.25) = 4.5$$

3.10 Level curves and alpha-cuts

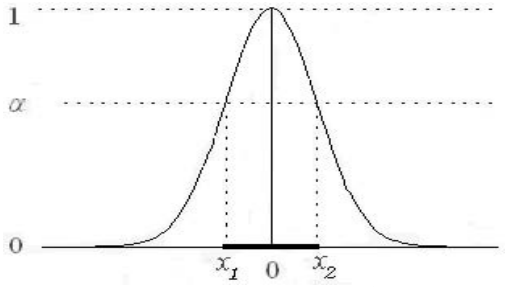
A **level set** (or **level curve**) of a function $A : X \to \mathbb{R}$ is the set $A^{-1}(\alpha)$ of points mapped by A to a constant α, or in other words, a set of the form

$$A^{-1}(\alpha) = \{x \in X : A(x) = \alpha\} \tag{3.9}$$

An α-**cut** for a function A is the set A_α of points for which the value of the function is greater than or equal to α — that is,

$$A_\alpha = \{x \in X : A(x) \geq \alpha\} \tag{3.10}$$

Example 3.14 For the Gaussian function $A(x) = e^{-\frac{x^2}{2}}$ the α-level set is the set $\{x_1, x_2\}$ such that $e^{-\frac{x_i^2}{2}} = \alpha$ for $0 < \alpha < 1$, the single point 0 for $\alpha = 1$, and empty for $\alpha = 0$.



The α-cut is the interval $[x_1, x_2]$ such that $e^{-\frac{x_i^2}{2}} = \alpha$ when $0 < \alpha < 1$, as depicted in the figure above, the single point 0 for $\alpha = 1$, and the entire domain for $\alpha = 0$.

The notion of α-cut is important in fuzzy set theory. The α-cut of a function $A : X \to [0, 1]$ is a (crisp) subset of X, and there is one such subset for each $\alpha \in [0, 1]$. A fundamental fact about the α-cuts A_α is that they determine A. This fact follows immediately from the equation

$$A^{-1}(\alpha) = A_\alpha \bigcap (\bigcup_{\beta > \alpha} A_\beta)' \tag{3.11}$$

This equation just says that the left side, $\{x : A(x) = \alpha\}$, namely those elements that A takes to α, is exactly the set of points that the two sets $\{x : A(x) \geq \alpha\}$ and $\{u : A(u) \not> \alpha\}$ have in common. But the two sets on the right are given strictly in terms of α-cuts, and knowing the sets $A^{-1}(\alpha)$ for all α determines A. So knowing all the α-cuts of A is the same as knowing A itself. We can state this as follows.

Theorem 3.6 *Let A and B be fuzzy sets. Then $A_\alpha = B_\alpha$ for all $\alpha \in [0, 1]$ if and only if $A = B$.*

3.10.1 Extension principle

When $f : X \to Y$ is a function between ordinary sets X and Y, and $A : X \to [0, 1]$ is a fuzzy subset of X, we can obtain a fuzzy subset B of Y by

$$B(y) = \bigvee_{f(x)=y} A(x) \tag{3.12}$$

This procedure is called the **extension principle**. The extension principle can be viewed as a function

$$Map(X,Y) \to Map(\mathcal{F}(X),\mathcal{F}(Y)) : f \mapsto E(f)$$

where $Map(X,Y)$ denotes the set of all functions from X to Y, $\mathcal{F}(X)$ and $\mathcal{F}(Y)$ denote the set of all fuzzy subsets of X and Y, respectively, $Map(\mathcal{F}(X),\mathcal{F}(Y))$ is the set of all functions from $\mathcal{F}(X)$ to $\mathcal{F}(Y)$, and $E(f)(A) = B$ as defined in Equation 3.12.[2]

The fuzzy set $E(f)(A) = B$ obtained above can be viewed as a composition of functions

$$B = \vee Af^{-1}$$

where

$$f^{-1}(y) = \{x \in X \mid f(y) = x\}$$

$\vee$ is the function from the set of subsets of $[0,1]$ to $[0,1]$ that takes a set to its supremum

$$\vee : 2^{[0,1]} \to [0,1] : S \mapsto \bigvee \{s : s \in S\}$$

and A is identified with the set function induced by A

$$A : 2^X \to 2^{[0,1]} : S \mapsto \{A(s) \mid s \in S\}$$

That is, we have the composition

$$B : Y \overset{f^{-1}}{\to} 2^X \overset{A}{\to} 2^{[0,1]} \overset{\vee}{\to} [0,1]$$

3.10.2 Images of alpha-level sets

Given a function $f : X \to Y$, there is a connection between the α-level sets, or the α-cuts, of a fuzzy set A and the fuzzy set $\vee Af^{-1}$. This relation is important in the calculus of fuzzy quantities. Note that the function f induces a partition of X into the subsets of the form $f^{-1}(y)$.

Theorem 3.7 *Let X and Y be sets, $A : X \to [0,1]$ and $f : X \to Y$. Then*

1. *$f(A_\alpha) \subseteq (\vee Af^{-1})_\alpha$ for all α.*
2. *$f(A_\alpha) = (\vee Af^{-1})_\alpha$ for $\alpha > 0$ if and only if for each $y \in Y$, $\vee A(f^{-1}(y)) \geq \alpha$ implies $A(x) \geq \alpha$ for some $x \in f^{-1}(y)$.*
3. *$f(A_\alpha) = (\vee Af^{-1})_\alpha$ for all $\alpha > 0$ if and only if for each $y \in Y$, $\vee A(f^{-1}(y)) = A(x)$ for some $x \in f^{-1}(y)$.*

[2] In appropriate categorical settings, the map E is functorial.

Proof. The theorem follows immediately from the equalities below.

$$\begin{aligned} f(A_\alpha) &= \{f(u) : A(x) \geq \alpha\} \\ &= \{y \in Y : A(x) \geq \alpha,\ f(u) = y\} \\ (\vee A f^{-1})_\alpha &= \{y \in Y : \vee A f^{-1}(y) \geq \alpha\} \\ &= \{y \in Y : \vee\{A(x) : f(x) = y\} \geq \alpha\} \end{aligned}$$

■

Of course, for some α, it may not be true that $\vee A(f^{-1}(y)) = \alpha$ for any y. The function $\vee A f^{-1}$ is sometimes written $f(A)$, and in this notation, the theorem relates $f(A_\alpha)$ and $f(A)_\alpha$.

The situation $X = X_1 \times X_2 \times \cdots \times X_n$ is of special interest. In that case, let $A^{(i)}$ be a fuzzy subset of X_i. Then $A^{(1)} \times \cdots \times A^{(n)}$ is a fuzzy subset of X, and trivially $\left(A^{(1)} \times \cdots \times A^{(n)}\right)_\alpha = A_\alpha^{(1)} \times ... \times A_\alpha^{(n)}$. The fuzzy subset $\vee(A^{(1)} \times \cdots \times A^{(n)})f^{-1}$ is sometimes written $f(A^{(1)}, ..., A^{(n)})$. In this notation, the third part of the theorem may be stated as

- $f(A_\alpha^{(1)}, ..., A_\alpha^{(n)}) = f(A^{(1)}, ..., A^{(n)})_\alpha$ for all $\alpha > 0$ if and only if for each $y \in Y$,
$$\vee(A^{(1)} \times \cdots \times A^{(n)})(f^{-1}(y))A^{(1)} \times \cdots \times A^{(n)})(x)$$
for some $x \in f^{-1}(y)$.

When $X = Y \times Y$, f is a binary operation on Y. If A and B are fuzzy subsets of Y and the binary operation is denoted $\circ$ and written in the usual way, then the theorem specifies exactly when $A_\alpha \circ B_\alpha = (A \circ B)_\alpha$, namely when certain supremums are realized.

3.11 Universal approximation

As we will see in the next two chapters, the design of fuzzy systems or of neural networks is aimed at approximating some idealistic input-output maps. As such, a general question is this. If $f : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary function, can we construct a fuzzy system or a neural network to approximate f? Of course, this question should be formulated in more specific terms. A positive answer to this question will put fuzzy and neural control on a firm theoretical basis, since it provides a guideline for design of successful controllers. Note however that if this is only an existence theorem, it does not actually tell us how to find the desired approximator for f.

A simple form in the theory of approximation of functions that is suitable for our purpose here is as follows. It is well-known that any continuous function $f : [a, b] \to \mathbb{R}$, defined on a closed and bounded (compact) interval $[a, b]$, can be approximated uniformly by polynomials — that is, for any degree of accuracy $\varepsilon > 0$, there is a polynomial $p(x)$ on $[a, b]$ such that $\sup_{x \in [a,b]} |p(x) - f(x)| < \varepsilon$. This is the classical Weierstrass theorem. The extension of this theorem, known

as the **Stone-Weierstrass theorem**, provides the most general framework for designing function approximators.

To formulate this theorem, we need some terminology and concepts. Let X be a set. A **distance** or **metric** d on X is a map $d : X \times X \to \mathbb{R}$ such that

1. $d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$.
2. $d(x, y) = d(y, x)$.
3. For any $x, y, z \in X$, $d(x, y) \leq d(x, z) + d(z, y)$.

On $\mathbb{R}$, absolute value $d(x, y) = |x - y|$ is a metric, and on the set $C([a, b])$ of all continuous real-valued functions defined on the closed interval $[a, b]$, the function

$$d(f, g) = \sup_{x \in [a,b]} |f(x) - g(x)|$$

for $f, g \in C([a, b])$ is also a metric. This metric is generated from the **sup-norm**

$$\|f\| = \sup_{x \in [a,b]} |f(x)|$$

by letting $d(f, g) = \|f - g\|$.

Since fuzzy systems or neural networks produce functions from $\mathbb{R}^n$ to $\mathbb{R}$, we need to specify the most general form of the counterpart in $\mathbb{R}^n$ of an interval $[a, b]$ in $\mathbb{R}$. So let (X, d) be a metric space, that is, a set X together with a metric d on it. For example, the Euclidean distance for $X = \mathbb{R}^n$ is

$$d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^2 \right)^{\frac{1}{2}} \quad \text{for } x = (x_1, ..., x_n) \text{ and } y = (y_1, ..., y_n)$$

A subset A of X is said to be **open** if for each $a \in A$ there is an $\varepsilon > 0$ such that $\{x \in X : d(x, a) < \varepsilon\} \subseteq A$. A subset B of X is said to be **closed** if its set complement $B' = \{x \in X : x \notin B\}$ is open. A subset A of X is **bounded** if $\sup\{d(x, y) : x, y \in A\} < +\infty$. On $\mathbb{R}^n$, subsets that are closed and bounded are called **compact subsets**. In metric spaces (X, d), a subset A of X is said to be **compact** if any open cover of it contains a finite subcover — that is, for any open sets A_i, $i \in I$, such that $A \subseteq \cup_{i \in I} A_i$, there is a finite set $J \subseteq I$ such that $A \subseteq \cup_{j \in J} A_j$. The closed intervals $[a, b]$ in $\mathbb{R}$ are compact, and this is the property that we need to generalize to $\mathbb{R}^n$.

Let (X, d) and (Y, e) be two metric spaces. A function $f : X \to Y$ is said to be **continuous at a point** $x \in S$ if for any $\varepsilon > 0$, there is $\delta > 0$ such that $e(f(x), f(y)) < \varepsilon$ whenever $d(x, y) < \delta$. The space $C([a, b])$ of continuous real-valued functions on $[a, b]$ is generalized to the space $C(X)$ of continuous real-valued functions on X, where (X, d) is a metric space and X is compact. The **sup-norm** on $C(X)$ is $\sup\{|f(x)| : x \in X\}$.

By exploiting the properties of polynomials on $[a, b]$, we arrive at the Stone-Weierstrass theorem. Its proof can be found in any text in Real Analysis, for example see [61].

Theorem 3.8 (Stone-Weierstrass theorem) *Let (X, d) be a compact metric space. Let $H \subseteq C(X)$ such that*

1. *H is a subalgebra of $C(X)$: If $a \in \mathbb{R}$ and $f, g \in H$, then $af \in H$ and $f + g \in H$.*
2. *H vanishes at no point of X: For any $x \in X$, there is an $h \in H$ such that $h(x) \neq 0$.*
3. *H separates points of X: If $x, y \in H$, $x \neq y$, then there is an $h \in H$ such that $h(x) \neq h(y)$.*

*Then H is **dense** in $C(X)$ — that is, for any $f \in C(x)$ and any $\varepsilon > 0$, there is an $h \in H$ such that $\|f - h\| < \varepsilon$, where $\|\cdot\|$ is the sup-norm on $C(X)$.*

This theorem says that one can approximate elements of $C(X)$ by elements of the subclass H arbitrarily closely. The application of this theorem to the setting of fuzzy and neural control is this. If our idealistic input-output map is a continuous function defined on a compact set of some metric space, then it is possible to approximate it to any degree of accuracy by fuzzy systems or neural networks, provided, of course, that we design our fuzzy systems or neural networks to satisfy the conditions of the Stone-Weierstrass theorem.

Even if we design fuzzy systems or neural networks to satisfy the conditions of the Stone-Weierstrass theorem, we still do not know which fuzzy system or which specific neural network architecture to choose as the desired approximator, since, as we already said, this theorem is only an existence theorem and is not a constructive one. Its usefulness is to guide us to set up approximation models that contain a "good" approximator. In real-world applications, how to find that good approximator is the main task. As we will see, in the context of fuzzy and neural control, this task is called **tuning**. Since it is possible to design fuzzy systems and neural networks to satisfy the conditions of the Stone-Weierstrass theorem, but fuzzy systems and neural networks can approximate arbitrary continuous functions defined on compact sets. So these two approaches possess the so-called **universal approximation property**.

3.12 Exercises and projects

1. For ordinary (crisp) subsets A and B of a set X, the intersection, union, and complement were defined in Equations 3.3 and 3.4. Prove that the intersection, union, and complement for ordinary sets $A, B : X \to \{0, 1\}$ satisfy all the following equations. (Each of these equations has been used as the basis for a generalization of intersection, union, and complement to fuzzy sets.)

   (a) intersection:

   i. $(A \cap B)(x) = A(x) \wedge B(x)$

ii. $(A \cap B)(x) = A(x)\,B(x)$
iii. $(A \cap B)(x) = \max\{A(x) + B(x) - 1, 0\}$

(b) union:

i. $(A \cup B)(x) = A(x) \vee B(x)$
ii. $(A \cup B)(x) = A(x) + B(x) - A(x)B(x)$
iii. $(A \cup B)(x) = \min\{A(x) + B(x), 1\}$

(c) complement:

i. $(X - A)(x) = 1 - A(x)$
ii. $(X - A)(x) = e^{-\frac{1}{\ln A(x)}}$
iii. $(X - A)(x) = (1 - A(x)^a)^{\frac{1}{a}}$ for $a > 0$
iv. $(X - A)(x) = \dfrac{1 - A(x)}{1 + (a-1)A(x)}$ for $a > 0$

2. Show that ordinary sets satisfy the De Morgan laws.

(a) $X - (A \cup B) = (X - A) \cap (X - B)$
(b) $X - (A \cap B) = (X - A) \cup (X - B)$

3. Let A and B be fuzzy subsets of X. Show that minimum and maximum, with complement $A'(x) = 1 - A(x)$, satisfy the De Morgan laws.

(a) $(A \vee B)'(x) = A'(x) \wedge B'(x)$ for all $x \in [0,1]$
(b) $(A \wedge B)'(x) = A'(x) \vee B'(x)$ for all $x \in [0,1]$

4. Let A and B be fuzzy subsets of X. Show that the algebraic product $(AB)(x) = A(x)\,B(x)$ and algebraic sum $(A \oplus B)(x) = A(x) + B(x) - AB(x)$, with the negation $A'(x) = 1 - A(x)$ satisfy the De Morgan laws:

(a) $(A \oplus B)'(x) = (A'B')(x)$ for all $x \in [0,1]$
(b) $(AB)'(x) = (A' \oplus B')(x)$ for all $x \in [0,1]$

5. Let A and B be fuzzy subsets of X. Show that the bounded product $(A \triangle B)(x) = (A(x) + B(x) - 1) \vee 0$ and bounded sum $(A \bigtriangledown B)(x) = (A(x) + B(x)) \wedge 1$, with the negation $A'(x) = 1 - A(x)$ satisfy the De Morgan laws:

(a) $(A \triangle B)'(x) = (A' \bigtriangledown B')(x)$ for all $x \in [0,1]$
(b) $(A \bigtriangledown B)'(x) = (A' \triangle B')(x)$ for all $x \in [0,1]$

6. Show that any t-norm $\circ$ satisfies the following.

(a) $x \circ y \leq x \wedge y$ for every $x, y \in [0,1]$
(b) $x \circ 0 = 0$ for every $x \in [0,1]$

7. Show that any t-conorm $*$ satisfies the following.

   (a) $x * y \geq x \vee y$ for every $x, y \in [0,1]$
   (b) $x * 1 = 1$ for every $x \in [0,1]$

8. Show that $\wedge$ is the only idempotent t-norm.

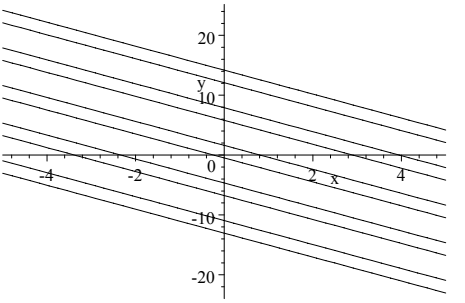
9. Show that the functions $\eta_\lambda(x) = \frac{1-x}{1+\lambda x}$ are negations for all $\lambda > -1$.

10. Prove that the equality

$$A^{-1}(\alpha) = A_\alpha \bigcap (\bigcup_{\beta > \alpha} A_\beta)'$$

given in Equation 3.11 always holds.

11. Explain why the 0.5-level set of $\sin(2x + y + 1)$ consists of all the lines $y = -2x - 1 + \frac{1}{6}\pi + 2n\pi$ and $y = -2x - 1 + \frac{5}{6}\pi + 2m\pi$ for integers m, n. Some of these lines are depicted in the following figure.



0.5-level set of $\sin t$

12. Suppose f is any function $\mathbb{R} \to \mathbb{R}$, and a, b, c are constants. Explain why the (nonempty) level sets for $f(ax + by + c)$ are families of straight lines.

13. Show that any α-cut of a fuzzy order ($\alpha > 0$) is an order, that is, show that the α-cut $S_\alpha = \{(x,y) \in X \times X : S(x,y) \geq \alpha\}$ satisfies

   (a) For all $x \in X$, $(x,x) \in S_\alpha$.
   (b) If $(x,y) \in S_\alpha$ and $(y,x) \in S_\alpha$ then $x = y$.
   (c) If $(x,y) \in S_\alpha$ and $(y,z) \in S_\alpha$ then $(x,z) \in S_\alpha$.

14. Let $A(x) = \begin{cases} x & \text{if } 0 \leq x \leq 1 \\ 2 - x & \text{if } 1 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$, $B(x) = \begin{cases} \frac{3x-1}{4} & \text{if } \frac{1}{3} \leq x \leq \frac{5}{3} \\ \frac{9-3x}{4} & \text{if } \frac{5}{3} \leq x \leq 3 \\ 0 & \text{otherwise} \end{cases}$,
$A'(x) = 1 - A(x)$ and $B'(x) = 1 - B(x)$ for $x \in [0,3]$. Show that

$$A(x) \wedge A'(x) = (A(x) \wedge A'(x) \wedge B(x)) \vee (A(x) \wedge A'(x) \wedge B'(x))$$

Find formulas for all the functions involved and also plot them.

15. Let $A(x) = \sin \pi x$, $B(x) = x^2$, $A'(x) = 1 - A(x)$, and $B'(x) = 1 - B(x)$ for $x \in [0,1]$. Show that

$$A(x) \wedge A'(x) = (A(x) \wedge A'(x) \wedge B(x)) \vee (A(x) \wedge A'(x) \wedge B'(x))$$

Find formulas for all the functions involved and also plot them.

16. If $\oplus$ is an averaging operator and f is a continuous, strictly increasing function $f : [0,1] \to [0,1]$ satisfying $f(0) = 0$ and $f(1) = 1$, show that

$$x \circledast y = f^{-1}(f(x) \oplus f(y))$$

is also an averaging operator.

17. Show that an OWA operator satisfies the three basic properties associated with an averaging operator.

18. Show that the function

$$t(x) = \max\left\{\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right\}$$

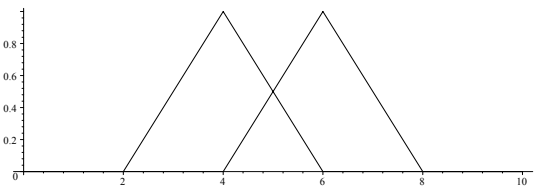
gives the triangular set determined by the points $(a,0)$, $(b,1)$, $(c,0)$ for $a < b < c$.

19. Show that the function

$$t(x) = \max\left\{\alpha \min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right\}$$

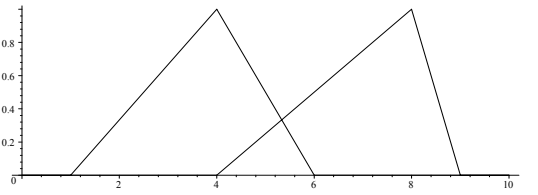
gives the trapezoidal set determined by the points $(a,0)$, (b,α), (c,α), $(d,0)$ for $a < b < c < d$ and $0 < \alpha \leq 1$.

20. Let $A[a,b,c]$ denote the triangular set determined by the points $(a,0)$, $(b,1)$, $(c,0)$. Using the fuzzy sets



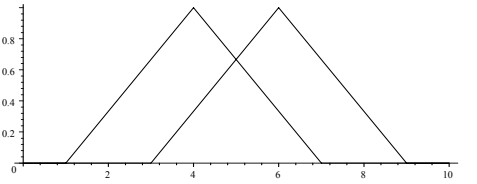
$A_1[2,4,6]$ and $A_2[4,6,8]$

and



$B_1\,[1,4,6]$ and $B_2\,[4,8,9]$

and



$C_1\,[1,4,7]$ and $C_2\,[3,6,9]$

graph the aggregated fuzzy set realized by the Mamdani method

$$O_{(5,5)}\,(z) = (A_1\,(5) \wedge B_1\,(5) \wedge C_1\,(z)) \vee (A_2\,(5) \wedge B_2\,(5) \wedge C_2\,(z))$$

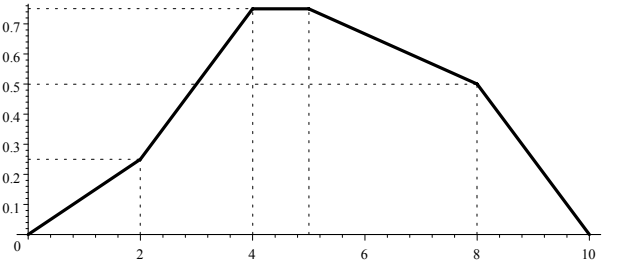
and defuzzify $O_{(5,5)}\,(z)$ by the "mean of maxima" method.

21. Repeat the previous exercise using the Larsen method, so that

$$O_{(5,5)}\,(z) = ((A_1\,(5) \wedge B_1\,(5)) \cdot C_1\,(z)) \vee ((A_2\,(5) \wedge B_2\,(5)) \cdot C_2\,(z))$$

and defuzzify by the "first of maxima" method.

22. Defuzzify the following set by each of the methods proposed in Section 3.9.



23. Repeat the previous exercise, but assume that the domain is the set of integers $1 \leq n \leq 10$.

Chapter 4

FUZZY CONTROL

Fuzzy logic controllers serve the same function as the more conventional controllers, but they manage complex control problems through heuristics and mathematical models provided by fuzzy logic, rather than via mathematical models provided by differential equations. This is particularly useful for controlling systems whose mathematical models are nonlinear or for which standard mathematical models are simply not available.

The implementations of fuzzy control are, in some sense, imitations of the control laws that humans use. Creating machines to emulate human expertise in control gives us a new way to design controllers for complex plants whose mathematical models are not easy to specify. The new design techniques are based on more general types of knowledge than differential equations describing the dynamics of the plants.

4.1 A fuzzy controller for an inverted pendulum

In Chapter 2, we presented examples of controlling an inverted pendulum by means of standard control. It is interesting to note that, with practice, humans can solve the inverted pendulum problem based solely on naive physics and common sense rather than on complicated mathematical models.[1] Balancing a broom upright in your hand, as in Figure 1.13 for example, involves more degrees of freedom and is inherently much more complicated than the inverted pendulum on a cart depicted in Figure 2.3. However, when you balance the broom, you are clearly not relying on a mathematical model that depends on accurate measurements of angles and velocity. For the inverted pendulum problem, it is possible to design a successful controller without knowing or using the plant dynamics. The implication is that, in a case where plant dynamics are not available, we can use this new technique to construct control laws.

[1] For a striking example, see the skilled performer Brad Byers balancing a 100 pound barbell on his chin, at http://www.bradbyers.com/balancer.html.

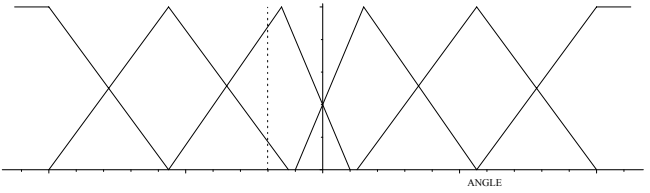
To balance an upright pendulum, we know from naive physics that the control force $u(t)$ should be chosen according to the magnitudes of the input variables $\theta(t)$ and $\theta'(t)$ that measure the angle from the upright position and the angular velocity. The relation between these variables is linguistic, a much weaker form than differential equations. That is exactly what happens in a human mind that processes information qualitatively. Humans choose $u(t)$ by using common sense knowledge in the form of "If... then..." rules, such as "If θ is very small and θ' is also small then u should be small," or "If the pendulum is in a balanced position, then hold very still, that is, do not apply any force." By taking all such rules into account, the inverted pendulum can be successfully controlled.

Now, in order to create an automatic control strategy duplicating human ability, it is necessary to be able to translate the above "If... then..." rules into some "soft" mathematical forms for machine processing. Looking at these "If... then..." rules, we ask ourselves questions like the following:

- Is "If... then..." an implication operator in logic?
- How can one model linguistic labels like "small," "medium," and "large"?
- Given a finite number of "If... then..." rules, how can we handle all possible numerical inputs that will be measured by machine sensors, in order to produce actual control actions?

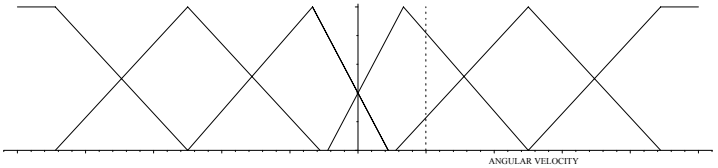
The answers to all these basic questions lie in fuzzy logic theory. The term "fuzzy control" refers to the science of building fuzzy controllers based on the mathematics of fuzzy logic.

Now let us look in more detail at the design of a fuzzy controller for the inverted pendulum on a cart. The input to the controller is the pair (θ, θ'). Let us take the input space to be $X \times Y$, where X and Y are intervals representing degrees for θ and degrees per second for θ'. The output or control space for u is an interval representing Newtons for u. The linguistic labels are modeled as fuzzy subsets of the spaces X, Y, and U by specifying their membership functions. For example, these linguistic labels could correspond to *negative big* (NB), *negative medium* (NM), *negative small* (NS), *positive small* (PS), *positive medium* (PM), and *positive big* (PB).



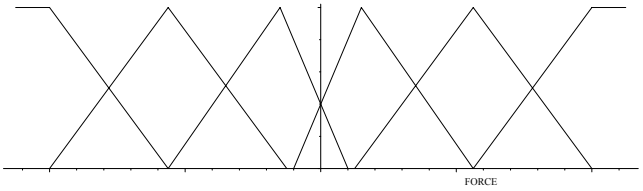
$A_1 = \text{NB}$ $A_2 = \text{NM}$ $A_3 = \text{NS}$ $A_4 = \text{PS}$ $A_5 = \text{PM}$ $A_6 = \text{PB}$

Fuzzy sets A_j for "angle"



$B_1 = \text{NB} \quad B_2 = \text{NM} \quad B_3 = \text{NS} \quad B_4 = \text{PS} \quad B_5 = \text{PM} \quad B_6 = \text{PB}$

Fuzzy sets B_j for "angular velocity"



$C_1 = \text{NB} \quad C_2 = \text{NM} \quad C_3 = \text{NS} \quad C_4 = \text{PS} \quad C_5 = \text{PM} \quad C_6 = \text{PB}$

Fuzzy sets C_j for "force"

The particular choice of membership functions chosen to represent the linguistic labels is somewhat arbitrary. One of the adjustments made during testing of the control system will be experimenting with different membership functions, that is, changing the parameters of these triangular functions. This process is called **tuning**.

The rules are then of the form

1. If *angle* is negative medium and *velocity* is positive small then *force* is negative small.

2. If *angle* is negative medium and *velocity* is positive medium then *force* is positive small.

3. If *angle* is negative small and *velocity* is positive medium then *force* is positive small.

and so forth. With six membership functions for each of X and Y, we can have 36 rules of the form

$$R_j : \textit{If } \theta \text{ is } A_j \text{ and } \theta' \text{ is } B_j \text{ then } u \text{ is } C_j$$

where A_j, B_j, and C_j are the fuzzy subsets of X, Y and U depicted above.

The look-up table

$\theta\backslash\theta'$	NB	NM	NS	PS	PM	PB
NB	NB	NB	NB	NM	NS	PS
NM	NB	NB	NM	NS	PS	PS
NS	NB	NM	NS	PS	PS	PM
PS	NM	NS	NS	PS	PM	PB
PM	NS	NS	PS	PM	PB	PB
PB	NS	PS	PM	PB	PB	PB

summarizes 36 possible rules in a compact form.

The input (θ, θ') to each rule R_j will result in a fuzzy subset of U. This fuzzy subset is often taken to be the minimum:

$$\varphi_j(u) = \min\{A_j(\theta), B_j(\theta'), C_j(u)\}$$

The fusion of rules, via the maximum, produces a fuzzy subset of U representing a control action:

$$\Psi(u) = \max\{\varphi_j(u) : j = 1, 2, ..., k\}$$

If, for example, the measurements are $\theta = -8^\circ$ and $\theta' = 2^\circ/\,\mathrm{s}$ then the fuzzy sets depicted above give the values

$$\begin{aligned} A_2(-8) &= 0.17 \\ A_3(-8) &= 0.88 \\ B_4(2) &= 0.6 \\ B_5(2) &= 0.82 \end{aligned}$$

and all other A_i and B_i are zero at this point. Thus the only rules pertinent for this input are the four

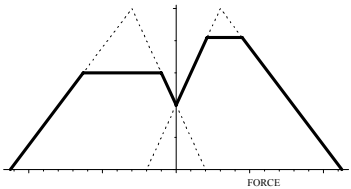
$\theta\backslash\theta'$	NM	NS
PS	NS	NS
PM	NS	PS

as all others give the value zero for this input. These four rules are said to "fire" for this input.

Combining these rules, we have the fuzzy set

$$\begin{aligned} \Psi(u) &= [A_2(-8) \wedge B_4(2) \wedge C_3(u)] \vee [A_2(-8) \wedge B_5(2) \wedge C_3(u)] \\ &\quad \vee [A_3(-8) \wedge B_4(2) \wedge C_3(u)] \vee [A_3(-8) \wedge B_5(2) \wedge C_4(u)] \\ &= [0.17 \wedge 0.6 \wedge C_3(u)] \vee [0.17 \wedge 0.82 \wedge C_3(u)] \\ &\quad \vee [0.88 \wedge 0.6 \wedge C_3(u)] \vee [0.88 \wedge 0.82 \wedge C_4(u)] \\ &= [0.6 \wedge C_3(u)] \vee [0.82 \wedge C_4(u)] \end{aligned}$$

and the aggregated output is the following fuzzy subset of U:



$$C_3 = \text{NS} \qquad C_4 = \text{PS}$$
$$\Psi(u) = [0.6 \wedge C_3(u)] \vee [0.82 \wedge C_4(u)]$$

To produce an actual control action for the input (θ, θ'), we need to summarize the fuzzy subset $\Psi(u)$ in some way. The control action depends on the defuzzification technique that is applied to this fuzzy set. A natural choice is the **centroid defuzzification method** discussed on page 120, namely, the actual control value is taken to be

$$u^* = \frac{\int u\Psi(u)du}{\int \Psi(u)du}$$

The control methodology described above, with sufficient tuning, will lead to successful control of the inverted pendulum.

The 36 possible rules determined by the look-up table on page 136 are far more than are needed. T. Yamakawa [84] successfully balanced an inverted pendulum on a cart using only seven rules. He used one additional fuzzy set, namely *approximately zero* (AZ), for each linguistic variable. The seven rules were as follows:

	θ	θ'	u
1	AZ	AZ	AZ
2	PS	PS	PS
3	PM	AZ	PM
4	PS	NS	AZ
5	NM	AZ	NM
6	NS	NS	NS
7	NS	PS	AZ

In practice, a small number of rules will often work as well as a larger number of rules. Working with a small number of rules has great advantages in terms of cost, efficiency, and speed of computing.

4.2 Main approaches to fuzzy control

The essential knowledge about the dynamics of a plant can be presented in the form of a linguistic rule base. In order to carry out a control synthesis, it is necessary to model the linguistic information as well as an inference process, an aggregation process, and possibly a defuzzification process. The use of fuzzy logic to transform linguistic knowledge into actual control laws leads to the field of fuzzy control. The main idea of fuzzy control is to formulate algorithms for

control laws using logical rules. That is why we sometimes call this methodology **fuzzy logic control**.

The rules are either obtained from experts (human controllers) or generated from numerical observations. When human experts experienced in controlling a given system are available, design engineers can interview these experts about their control strategies and then express the experts' knowledge as a collection of control "If...then..." rules. Even when the input-output relations in a complex system are not known precisely, the behavior of the control process provides a valuable source of information for suggesting control strategies and extracting fuzzy rules. That is, one can supply numerical inputs and observe numerical outputs of a control system as a means to obtain linguistic control "If...then..." rules.

It is clear that there is flexibility in choices of membership functions, rules, fuzzy logic operations, and defuzzification procedures when designing fuzzy controllers. Note that fuzzy systems, or fuzzy models, are actually mathematical models — that is, they are mathematical descriptions of relations between variables, using membership functions of fuzzy sets. These mathematical models are flexible, but they are not "fuzzy" in layman terms. The membership functions themselves are precise mathematical functions.

Just as in the case of standard control, where existence of control laws should be guaranteed before proceeding to obtain the control algorithm, we should ask whether a given linguistic rule base is sufficient to derive a "successful" control law. The rule base is a description of how we should control the plant. Each rule is *local* in nature — that is, each rule tells us how we should control the plant in some small region of the input space X. Since we are going to use this information to derive a *global* control law φ, a map on the entire input space X, these small regions specified in the rules should cover all points of X in some fashion. In other words, the membership functions defined on X should form a fuzzy partition of X.

The methodology of fuzzy control consists of selecting and using
1. a collection of rules that describe the control strategy,
2. membership functions for the linguistic labels in the rules,
3. logical connectives for fuzzy relations, and
4. a defuzzification method.

At the end, the derived control law is the realization of a function φ from X to U, the space of control variables.

The strategy above for deriving control laws is based somewhat on common sense. However, its successful applications can be explained theoretically in the context of the theory of approximation of functions. See the statement of the Stone-Weierstrass theorem, Chapter 3, theorem 3.8, for example. Fuzzy systems satisfy a universal approximation property [78], which means that there is a system that will do the task you want to accomplish. However, this does not say how to set up or tune such a fuzzy system. Such mathematical justifications only provide some assurance that, if you work at it hard enough, it is possible to design successful fuzzy controllers. The capability of fuzzy controllers can

also be assessed by using a concept known as *Vapnik-Chervonenki dimension* of a class of functions, just as in the case of neural networks [23]. This is discussed in Section 5.3.

We will discuss some well-known methods for constructing fuzzy control systems. We look first at the Mamdani and Larsen methods.

4.2.1 Mamdani and Larsen methods

The first application of fuzzy logic was developed by E. H. Mamdani in 1974. He designed an experimental fuzzy control system for a steam engine and boiler combination by synthesizing a set of linguistic control rules obtained from experienced human operators. The popular method known today as the Mamdani method is very similar to his original design.

For the Mamdani and Larsen methods, a typical rule base is of the form

$$R_j\text{: If } \mathbf{x} \text{ is } A_j \text{ then } u \text{ is } B_j,\ j = 1, 2, ..., r$$

where $\mathbf{x} = (x_1, ..., x_n) \in X$ and $u \in U$, and $A_j(\mathbf{x}) = \min_{i=1...n} \{A_{ji}(x_i)\}$ for

$$\begin{aligned} A_j &= A_{1j} \times A_{2j} \times \cdots \times A_{nj} : X = X_1 \times X_2 \times \cdots \times X_n \to [0,1] \\ B_j &: U \to [0,1] \end{aligned}$$

How can we interpret the rule "R_j: If $A_j(x)$ then $B_j(u)$"? One common view is that each rule R_j represents a fuzzy relation on $X \times U$. In other words, when the input is x, then the degree to which a value $u \in U$ is consistent (or compatible) with the meaning of R_j is

$$C_j(x, u) = T(A_j(x), B_j(u))$$

where T is some t-norm. Thus, C_j is a fuzzy subset of $X \times U$. For a given input x, C_j induces a fuzzy subset of U, namely

$$C_j^x(x) : u \longrightarrow C_j(x, u),\ u \in U,\ j = 1, 2, ..., N$$

Thus, the output of each rule R_j is a fuzzy subset of U. The Mamdani method uses minimum for the t-norm

$$C_j^x(u) = C_j(x, u) = A_j(x) \wedge B_j(u) \tag{4.1}$$

and the Larsen method uses the product for the t-norm

$$C_j^x(u) = C_j(x, u) = A_j(x) B_j(u) \tag{4.2}$$

Having translated each rule R_j into C_j^x, the next task is to fuse all the rules together. We face the following problem: Given N fuzzy subsets C_1^x,...,C_N^x of U, we need to combine them to produce an overall output. From a semantic viewpoint, of course, it is a question of how the rules are connected. From a mathematical viewpoint, we want to form a single fuzzy subset of U from the

C_j^x, $j = 1, ..., N$. In the Mamdani and Larsen methods, this is done by taking the maximum. In the Mamdani method, for each $\mathbf{x} \in X = X_1 \times \cdots \times X_n$, this gives the fuzzy subset

$$C^x(u) = C(u|x) = \max_{j=1,...,N} \left(\min_{i=1,...,n} \{A_{ji}(x_i), B_j(u)\} \right) \tag{4.3}$$

of U. Equation 4.3 is called **Mamdani synthesis**.

In the Larsen method, for each $\mathbf{x} \in X$, this gives the fuzzy subset

$$C^x(u) = C(u|x) = \max_{j=1,...,N} \left(\left(\min_{i=1,...,n} \{A_{ji}(x_i)\} \cdot B_j(u) \right) \right) \tag{4.4}$$

of U. Equation 4.4 is called **Larsen synthesis**.

The overall output is, for each x, a fuzzy subset C^x of U. What is the meaning of C^x? When the input x is given, each control action u is compatible with degree $C^x(u)$. We need a single numerical output $u^* = u^*(x)$ for the control law, and we need to get this from the fuzzy subset C^x. In other words, we need to *defuzzify* the fuzzy subset C^x. In the Mamdani and Larsen approaches, we defuzzify C^x by using the center of area (center of gravity or centroid) procedure, namely

$$u^*(x) = \frac{\int_U uC^x(u)\,du}{\int_U C^x(u)\,du}$$

Of course, there are many different ways to defuzzify C^x. See Section 3.9 for a discussion of defuzzification methods. The complexity, or speed, of the computation will be affected by the method of defuzzification. The question of how to choose a defuzzification method should be settled by the performance of the induced controller.

4.2.2 Model-based fuzzy control

Another useful approach to fuzzy control is a functional form of fuzzy system, due to Takagi and Sugeno, in which defuzzification is not needed. This essentially model-based fuzzy control method can be used when it is possible to describe the dynamics of a plant locally in approximate terms.

The rule base in this case is of the form

$$R_j : \text{If } x_1 \text{ is } A_{j1} \text{ and } ... \text{ and } x_n \text{ is } A_{jn} \text{ then } u_j = f_j(x_1, x_2, ..., x_n)$$

for $j = 1, 2, ..., r$, where the x_i are observed values of the input variables, the f_j are functions, and the A_{ij} form a fuzzy partition of the input space. Taking the product of the A_{ji}'s we can express the rules in the simpler form

$$R_j : \text{“If } \mathbf{x} \text{ is } A_j \text{ then } u_j = f_j(x_1, x_2, ..., x_n)\text{”}$$

The **firing degree** of each rule R_j is $A_j(\mathbf{x})$, and the overall output control value is taken to be

$$u(\mathbf{x}) = \sum_{j=1}^{r} A_j(\mathbf{x}) f_j(\mathbf{x}) \Big/ \sum_{j=1}^{r} A_j(\mathbf{x})$$

Note that the premise is in the same form as for the Mamdani and Larsen methods, but the consequent is of a different form.

In the Takagi-Sugeno method each f_j is a linear function

$$f_j(x_1, ..., x_n) = a_{0j} + \sum_{i=1}^{n} \alpha_{ij} x_i$$

Other forms in common use are quadratic

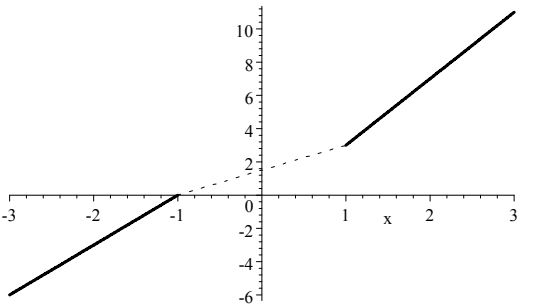
$$f_j(x_1, ..., x_n) = a_{0j} + \sum_{i=1}^{n} \alpha_{ij} x_i^2$$

and trigonometric

$$f_j(x_1, ..., x_n) = \exp\left(\sum_{i=1}^{n} \alpha_{ij} \sin x_i\right)$$

The choice of the consequents $f_j(\mathbf{x})$ depends upon the particular application.

Example 4.1 The Takagi-Sugeno rules provide a means to interpolate between piecewise linear mappings. For the partial mapping



$$y = \begin{cases} 3 + 3x \text{ for } x \leq -1 \\ -1 + 4x \text{ for } x \geq 1 \end{cases}$$

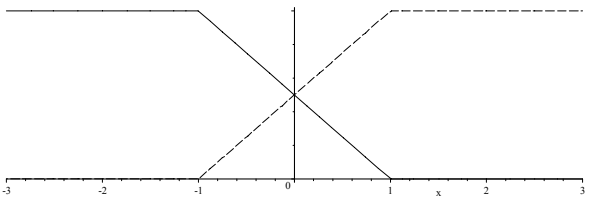
take two fuzzy sets

$$A_1(x) = \begin{cases} 1 & \text{if} \quad x \leq -1 \\ \frac{1-x}{2} & \text{if} \quad -1 \leq x \leq 1 \\ 0 & \text{if} \quad 1 \leq x \end{cases}$$

and

$$A_2(x) = \begin{cases} 0 & \text{if} \quad x \leq -1 \\ \frac{1+x}{2} & \text{if} \quad -1 \leq x \leq 1 \\ 1 & \text{if} \quad 1 \leq x \end{cases}$$

each having value 1 on one of the pieces of the mapping and 0 on the other.



$A_1(x)$ solid line, $A_2(x)$ dashed line

and rules

$$R_1: \text{If } x \text{ is } A_1 \text{ then } f_1(x) = 1 + x$$
$$R_2: \text{If } x \text{ is } A_2 \text{ then } f_2(x) = 2 + x$$

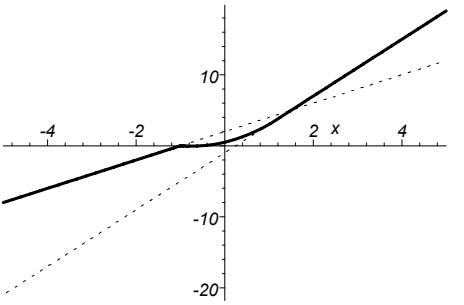
Now

$$A_1(x) f_1(x) + A_2(x) f_2(x) = \begin{cases} 2 + 2x & \text{if} \quad x \leq -1 \\ \frac{1}{2} + \frac{3}{2}x + x^2 & \text{if} \quad -1 \leq x \leq 1 \\ -1 + 4x & \text{if} \quad 1 \leq x \end{cases}$$

and

$$A_1(x) + A_2(x) = 1$$

giving the following plot:



$$y = \sum_{j=1}^{2} A_j(x) f_j(x_j) \Big/ \sum_{j=1}^{2} A_j(x)$$

These models can also be used as a nonlinear interpolator between linear systems. Sugeno proposes rules of the form

$$R_i\text{: If } z_1 \text{ is } C_{i1} \text{ and} \ldots \text{and } z_p \text{ is } C_{ip} \text{ then } \dot{\mathbf{x}}_i(t) = A_i(\mathbf{x}(t)) + B_i \mathbf{u}((t)) \quad (4.5)$$

for $i = 1, 2, ..., r$. In this rule, Sugeno uses a canonical realization of the system known in classical control theory as the "controller canonical form." Here, $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$ is the n-dimensional state vector, $\mathbf{u}(t) = (u_1(t), \ldots, u_m(t))$ is the m-dimensional input vector, A_i, B_i, $i = 1, 2, ..., r$,

are state and input matrices, respectively, and $\mathbf{z}(t) = (z_1(t), \ldots, z_p(t))$ is the p-dimensional input to the fuzzy system. The output is

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \frac{\sum_{i=1}^{r}\left[A_i\mathbf{x}(t) + B_i\mathbf{u}(t)\right]\tau_i(\mathbf{z}(t))}{\sum_{i=1}^{r}\tau_i(\mathbf{z}(t))} \\ &= \frac{\sum_{i=1}^{r}A_i\tau_i(\mathbf{z}(t))}{\sum_{i=1}^{r}\tau_i(\mathbf{z}(t))}\mathbf{x}(t) + \frac{\sum_{i=1}^{r}B_i\tau_i(\mathbf{z}(t))}{\sum_{i=1}^{r}\tau_i(\mathbf{z}(t))}\mathbf{u}(t)\end{aligned}$$

where

$$\tau_i(\mathbf{z}(t)) = \triangle_{k=1}^{p}C_{ik}(z_k(t))$$

for some appropriate t-norm $\triangle$. In the special case $r = 1$, the antecedent is a standard linear system:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t)$$

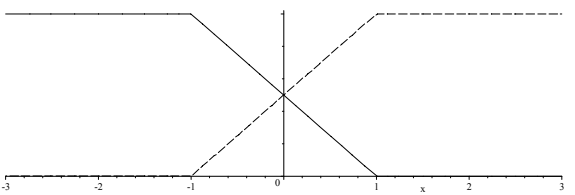
In the general case, such a fuzzy system can be thought of as a nonlinear interpolator between r linear systems.

Example 4.2 Suppose that $z(t) = x(t)$, $p = n = m = 1$, and $r = 2$. Take the two fuzzy sets

$$C_1(z) = \begin{cases} 1 & if \quad z \leq -1 \\ \frac{1-z}{2} & if \quad -1 \leq z \leq 1 \\ 0 & if \quad 1 \leq z \end{cases}$$

and

$$C_2(z) = \begin{cases} 0 & if \quad z \leq 0 \\ \frac{1+z}{2} & if \quad -1 \leq z \leq 1 \\ 1 & if \quad 1 \leq z \end{cases}$$



$C_1(z)$ *solid line,* $C_2(z)$ *dashed line*

and rules

$$\begin{aligned} &R_1\text{: } \textit{If } z \textit{ is } C_1 \textit{ then } \dot{x}_1 = -x_1 + 2u_1 \\ &R_2\text{: } \textit{If } z \textit{ is } C_2 \textit{ then } \dot{x}_2 = -2x_2 + u_2 \end{aligned}$$

so $A_1 = -1$, $B_1 = 2$, $A_2 = -2$, and $B_2 = 1$. Since $\tau_1(z) + \tau_1(z) = C_1(z) + C_2(z) = 1$, the output of the system is

$$\dot{\mathbf{x}} = (-C_1(z(t)) - C_2(z(t)))\mathbf{x} + (2C_1(z(t)) + C_2(z(t)))\mathbf{u}$$

Thus, we have

$$\dot{\mathbf{x}} = \begin{cases} -\mathbf{x} + 2\mathbf{u} & if \quad z \leq -1 \\ -\mathbf{x} + \left(\frac{3-z}{2}\right)\mathbf{u} & if \quad -1 \leq z \leq 1 \\ -2\mathbf{x} + \mathbf{u} & if \quad 1 \leq z \end{cases}$$

4.3 Stability of fuzzy control systems

Fuzzy control designs seem appropriate for nonlinear systems whose mathematical models are not available. While the design of fuzzy controllers, based mostly on linguistic rules, is relatively simple and leads to successful applications; it remains to be seen how the most important problem in analysis of control systems, namely the problem of stability, is addressed.

In standard control theory, one needs to study the stability of the control system under consideration, either for its open-loop dynamics or its closed-loop dynamics (assuming a control law has been designed). This is based on the availability of the mathematical model for the system. Although in nonlinear systems where solutions of nonlinear differential equations are difficult to obtain, the knowledge of these equations is essential for stability analysis, such as in the method of Lyapunov.

From a practical viewpoint, one can always use simulations to test stability of the system when a fuzzy controller has been designed. But it is desirable to have a theoretical analysis of stability rather than just "blind" simulations. The problem of stability analysis of fuzzy control systems is still an active research area in the theory of fuzzy control. Now, as stated above, it is difficult to see how stability analysis can be carried out without mathematical models of the systems, but fuzzy control is intended primarily for systems whose mathematical models are not available in the first place. Thus, to have a stability analysis for fuzzy control, it seems plausible that some form of models should be known. In other words, stability analysis of fuzzy control systems is only possible if the problem is model-based.

A model-based approach to fuzzy control does not necessarily mean that mathematical models are available. The models we refer to are fuzzy models as opposed to precise mathematical models used in standard control. Mathematically speaking, a fuzzy model is a mathematical model that involves fuzzy terms. Thus, fuzzy models are generalizations of "crisp" models.

The stability analysis of model-based fuzzy control systems was carried out by Tanaka and Sugeno [73] in 1992 using the Takagi-Sugeno model [72]. Here is a summary of their approach to stability analysis, using Lyapunov's direct method. In the discrete form, the Takagi-Sugeno fuzzy dynamical model describes locally a linear relationship between inputs and outputs of the system.

Let the state-variable and control input be

$$\begin{aligned} \mathbf{x}(k) &= \begin{pmatrix} x_1(k) & x_2(k) & \cdots & x_n(k) \end{pmatrix}^T \\ \mathbf{u}(k) &= \begin{pmatrix} u_1(k) & u_2(k) & \cdots & u_n(k) \end{pmatrix}^T \end{aligned}$$

The fuzzy model is a collection of fuzzy rules R_j, $j = 1, 2, ..., r$, of the form

$$\begin{aligned} R_j\text{: If } & x_1(k) \text{ is } A_{j1} \text{ and } x_2(k) \text{ is } A_{j2} \text{ and ... and } x_n(k) \text{ is } A_{jn}, \\ & \text{then } \mathbf{x}(k+1) = A_j\mathbf{x}(k) + B_j\mathbf{u}(k) \end{aligned}$$

where the A_{ji}'s, $j = 1, ..., r$; $i = 1, ..., n$, are fuzzy subsets of the state space.

For a given pair $(\mathbf{x}(k), \mathbf{u}(k))$, the next state $\mathbf{x}(k+1)$ is obtained from the above rule base as follows: Let $A_{ji}(x_i(k)) = w_{ji}(k)$ be the membership degree of $x_i(k)$ in the fuzzy set A_{ji}. Using the product t-norm for fuzzy intersection ("and"), the membership degree of $\mathbf{x}(k)$ in rule R_j is

$$v_j(k) = \prod_{i=1}^{n} w_{ji}(k)$$

Then

$$\mathbf{x}(k+1) = \frac{\sum_{j=1}^{r} v_j(k) [A_j x(k) + B_j u(k)]}{\sum_{j=1}^{r} v_j(k)} \tag{4.6}$$

The Takagi-Sugeno fuzzy dynamical model is specified by the matrices A_j, $j = 1, 2, ..., r$ and the fuzzy sets A_{ji}, $i = 1, 2, ..., n$; $j = 1, 2, ..., r$.

The open-loop system (no control input u) is

$$\mathbf{x}(k+1) = \frac{\sum_{j=1}^{r} v_j(k) A_j x(k)}{\sum_{j=1}^{r} v_j(k)} \tag{4.7}$$

which is nonlinear.

Note that it suffices to study the stability of (4.7) since once a control law $u(\cdot)$ is designed, the closed-loop system will be of the same form. Recall that the equilibrium of the system (4.7) is asymptotically stable in the large if $x(k) \to 0$ as $k \to +\infty$, for any initial $x(0)$.

Theorem 4.1 (Tanaka and Sugeno) *A sufficient condition for the equilibrium of the system (4.7) to be asymptotically stable in the large is the existence of a common positive definite matrix P such that $A_j^T P A_j < 0$ for all $j = 1, 2, ..., r$ (where A_j^T denotes the transpose of A_j).*

Proof. Since (4.7) is nonlinear, the theorem will be proved if we can find a Lyapunov function for the system, that is, a scalar function $V(\mathbf{x}(k))$ such that

1. $V(0) = 0$
2. $V(\mathbf{x}(k)) > 0$ for $\mathbf{x}(k) \neq 0$
3. $V(\mathbf{x}(k)) \to \infty$ as $\|\mathbf{x}(k)\| \to \infty$
4. $\triangle V(\mathbf{x}(k)) = V(\mathbf{x}(k+1)) - V(\mathbf{x}(k)) < 0$

Consider the function V defined by

$$V(\mathbf{x}(k)) = \mathbf{x}(k)^T P \mathbf{x}(k)$$

Then (1), (2), and (3) are easy to verify. For (4), we have

$$\begin{aligned}\triangle V(\mathbf{x}(k)) &= \frac{1}{\sum_{j=1}^{r}\sum_{i=1}^{r} v_j(k) v_i(k)} \left[\sum_{j=1}^{r} v_j^2(k) \mathbf{x}(k)^T \left[A_j^T P A_j - P\right] \mathbf{x}(k) \right. \\ &\left. + \sum_{1 \le i < j \le r} v_i(k) v_j(k) \mathbf{x}(k)^T \left[A_i^T P A_j + A_j^T P A_i - 2P\right] \mathbf{x}(k) \right]\end{aligned}$$

Then, in view of the hypothesis and the fact that if A, B, P are $n \times n$ matrices with P positive definite and both $A^T PA - P < 0$ and $B^T PB - P < 0$, then

$$A^T PA + B^T PB - 2P < 0$$

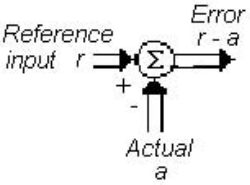
we see that $\triangle V(\mathbf{x}(k)) < 0$. ■

4.4 Fuzzy controller design

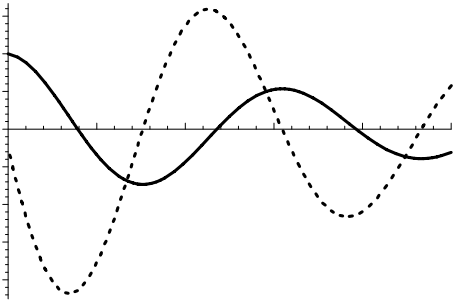
Let us now investigate the ease with which we can design a fuzzy controller. In this section, we give three examples of controller design using fuzzy methodology. Each of these examples can be compared with an example given in Chapter 2 of controller design using standard methodology to solve a similar problem.

4.4.1 Example: automobile cruise control

In Section 2.7.1, we discussed the design of an automobile cruise control system using the standard approach. Here we solve the same problem using the Mamdani method, a fuzzy approach (see pages 110 and 140). It should become clear that this fuzzy approach, which provides a model-free approach to developing a controller, is simpler.



(a) Velocity error



(b) Velocity error (solid), acceleration (dotted)

Figure 4.1. Velocity error and its rate of change

We define **velocity error** as the difference "desired velocity minus actual velocity" where the desired velocity is the set-point. Then the acceleration is the time-derivative of the velocity error. These are represented in Figure 4.1 where the velocity error varies between positive (the automobile is traveling too slowly) and negative (the automobile is traveling too fast) and the rate of change of the velocity error varies between negative and positive.

From the dynamics of the system, in this case an automobile, if the actual velocity is less than the desired velocity, we need to accelerate the vehicle by

providing as much force as needed to bring the velocity to the desired level. If the actual velocity is more than the desired velocity, we need to reduce the force so that the vehicle can decelerate and attain the desired velocity. If the vehicle is at the desired velocity, we apply sufficient force to maintain the velocity at the set-point. This knowledge of the system behavior allows us to formulate a set of general rules. Following are some example rules:

- If velocity error is positive and acceleration is negative then apply maximum force.
- If velocity error is negative and acceleration is positive then apply minimum force.
- If velocity error is zero and acceleration is zero then apply normal force.

The fuzzy controller we wish to design would require two inputs, namely velocity error and acceleration, in order to generate the appropriate output, namely engine force. Schematically, this is illustrated in Figure 4.2.

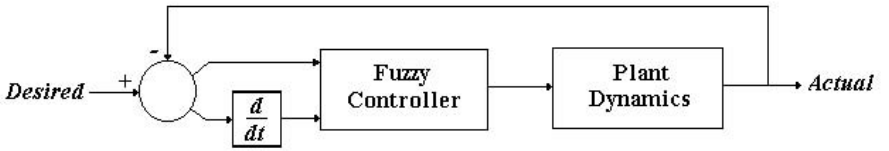
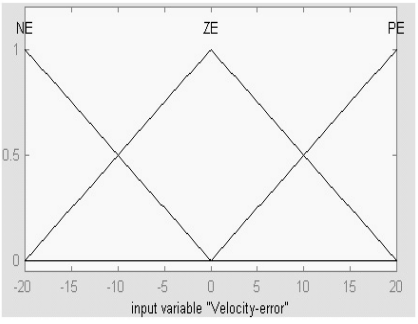
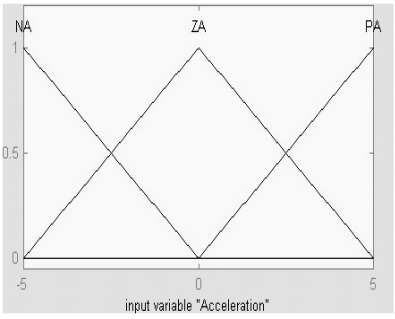


Figure 4.2. Design of fuzzy controller

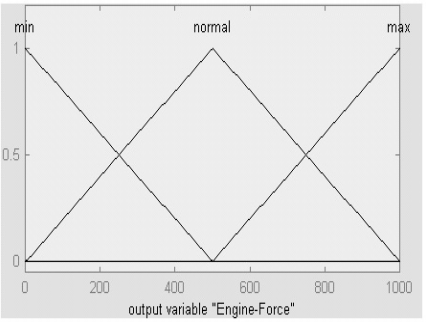
Using the MATLAB Fuzzy Toolbox, we can set up prototype triangular fuzzy sets for the fuzzy variables, namely, velocity error, acceleration, and engine force. Type *fuzzy* at the MATLAB prompt to bring up the fuzzy inference system (FIS) editor. The result is illustrated in Figure 4.3 (a), (b), and (c).





(a) Velocity error (b) Acceleration

Figure 4.3. Prototype membership functions for automobile cruise control



(c) Engine force

Figure 4.3. Prototype membership functions for automobile cruise control

With the fuzzy sets defined, it is now possible to associate the fuzzy sets in the form of fuzzy rules. When there are two inputs and one output, it is easy to visualize the associations in the form of a table, called a **fuzzy associative memory** or **decision table**. For this system, we can obtain such a table as shown in Figure 4.4.

Velocity-Error \ *Acceleration*	*NA*	*ZA*	*PA*
NE	*Min*	*Min*	*Min*
ZE	*Normal*	*Normal*	*Normal*
PE	*Max*	*Max*	*Max*

Figure 4.4. Fuzzy associative memory table for automobile cruise control

The associative memory is a set of nine rules of the form:

1. If velocity error is NE and acceleration is NA, then engine force is Min.
2. If velocity error is NE and acceleration is ZE, then engine force is Min.
3. If velocity error is NE and acceleration is PA, then engine force is Min.
4. If velocity error is ZE and acceleration is NA, then engine force is Normal.
5. If velocity error is ZE and acceleration is ZA, then engine force is Normal.
6. If velocity error is ZE and acceleration is PA, then engine force is Normal.

7. If velocity error is PE and acceleration is NA, then engine force is Max.

8. If velocity error is PE and acceleration is ZA, then engine force is Max.

9. If velocity error is PE and acceleration is PA, then engine force is Max.

Using MATLAB the fuzzy control surface can be developed as shown in Figure 4.5. This **control surface** is the output plotted against the two inputs, and displays the range of possible defuzzified values for all possible inputs. It is an interpolation of the effects of the nine rules.

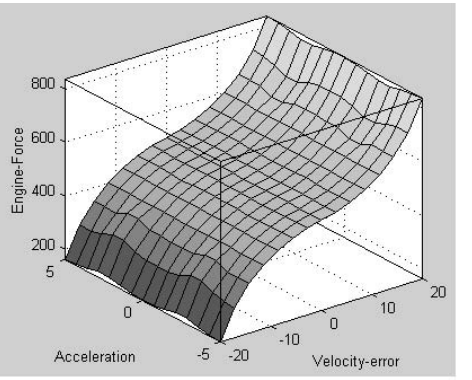


Figure 4.5. Control surface

Note that the control surface for a simple PD controller is a plane, which could approximate this fuzzy control surface only in a small neighborhood of the origin.

The system can then be simulated using the Simulink package with the system modeled as shown in Figure 4.6.

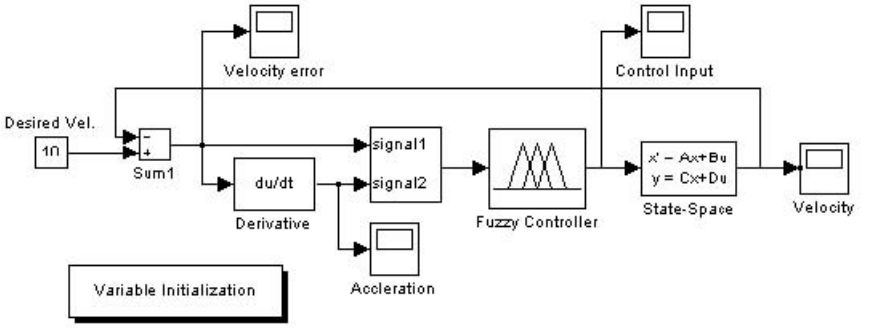


Figure 4.6. Simulink diagram for automobile cruise control using a state-space model of the automobile

We describe briefly how to set up a simulation using the Simulink package in MATLAB. Simulink allows dynamic elements of a system to be connected using both linear and nonlinear elements. Referring to Figure 4.6, the desired

velocity is set up as a constant that feeds a signal to a summer. Since this is a reference signal, we designate the signal input as positive. This signal is summed with the output of the plant model that is designated as a negative input to the summer. The resulting error signal serves two purposes, namely, it gives clear indication whether the plant output is above or below the desired value (10 meters/second in this case), and allows us to compute the derivative of the error to determine if the error is increasing or decreasing in time. Both the error signal and its derivative are fed to a multiplexer (Mux) that provides input to the fuzzy controller. The output signal of the fuzzy controller is then fed to a state-space model of the system. MATLAB provides a convenient way to set up the elements of the state-space model. All signals desired as outputs can be monitored using a variety of output devices. For convenience, it is desirable to provide a "Variable Initialization" block so that all variables at the beginning of a simulation can automatically be set to zero. This only assures that we are starting the simulation with the plant at rest. In cases where there are multiple outputs from the controller and/or the plant, we will need a demultiplexer (Demux) to channel the vector of outputs to the appropriate devices.

The scope outputs for velocity error, acceleration, fuzzy controller output, and actual velocity are depicted in Figure 4.7.

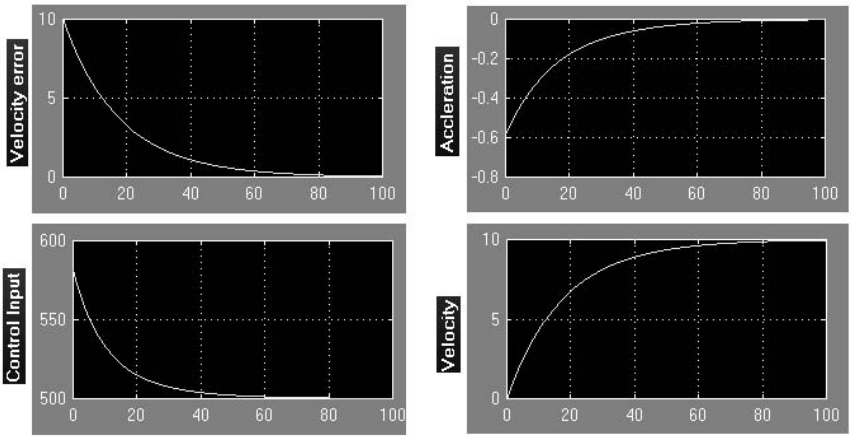


Figure 4.7. Simulated output of automobile cruise control

It is clear from the simulated outputs that rapid prototyping of the controller is possible. However, the membership functions will have to be "tuned" to yield the desired output characteristics — that is, the parameters of these functions will need to be adjusted. When the membership functions are tuned, this will change the shape of the control surface and affect the behavior of the closed-loop control system. Some tuning techniques are included in the following example.

4.4.2 Example: controlling dynamics of a servomotor

We now investigate the design of a fuzzy controller to control the dynamics of a DC servomotor. Compare this with the example of a classical controller design to solve this problem (see page 71).

In this example, we wish to control the angular position of the rotor to coincide with some desired position. Let us define position error as the difference "desired position minus actual position" where desired position is the set point. We can then use position error and rotor velocity as the two input variables to the controller and obtain the necessary voltage to be applied to the motor as the output of the controller. Schematically, the fuzzy controller can be shown in Figure 4.8.

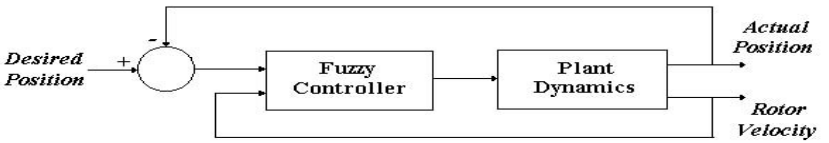
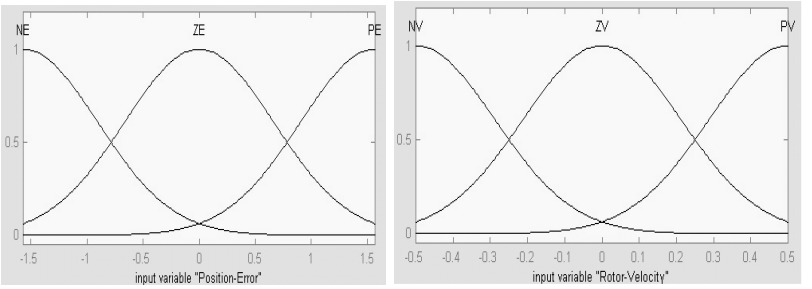


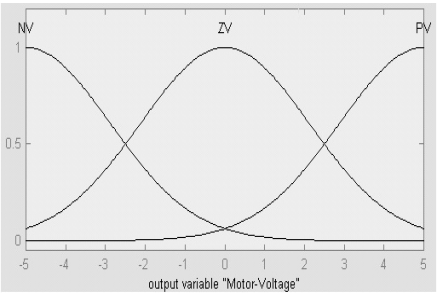
Figure 4.8. Closed-loop control system

The dynamics of the DC servomotor are such that we can rotate the shaft of the motor either in the clockwise or counterclockwise direction by simply reversing the polarity of the applied voltage. Therefore, if the actual position is less than the desired position (a positive error), we can achieve the desired position faster by boosting the voltage applied to the motor terminals. For a negative error, when the rotor position is greater than that which is desired, the voltage polarity can be reversed so that the rotor turns in the opposite direction.



(a) Position error (b) Rotor velocity

Figure 4.9. Prototype fuzzy membership functions for DC servomotor control



(c) Motor voltage

Figure 4.9. Prototype fuzzy membership functions for DC servomotor control

This knowledge of the system behavior allows us to formulate a set of general rules as, for example,

1. If position error is positive and velocity is negative, then apply positive voltage.

2. If position error is negative and velocity is positive, then apply negative voltage.

3. If position error is zero and velocity is zero, then apply zero voltage.

Using the MATLAB fuzzy toolbox, we can set up prototype Gaussian fuzzy sets for the fuzzy variables, namely, position error, velocity, and motor voltage. This is illustrated in Figure 4.9.

With the fuzzy sets defined, it is now possible to associate the fuzzy sets in the form of fuzzy rules. When there are two inputs and one output, it is easy to visualize the associations in the form of a matrix. For this system, we can obtain the matrix shown in Figure 4.10.

Position-Error \ *Error-Velocity*	*NV*	*ZV*	*PV*
NE	*Negative Voltage*	*Negative Voltage*	*Negative Voltage*
ZE	*Negative Voltage*	*Zero Voltage*	*Positive Voltage*
PE	*Positive Voltage*	*Positive Voltage*	*Positive Voltage*

Figure 4.10. Fuzzy associative matrix for DC servomotor control

The matrix in Figure 4.10 summarizes the following set of nine rules:

1. If position error is NE and velocity is NA, then motor voltage is Negative.
2. If position error is NE and velocity is ZE, then motor voltage is Negative.
3. If position error is NE and velocity is PA, then motor voltage is Negative.
4. If position error is ZE and velocity is NA, then motor voltage is Zero.
5. If position error is ZE and velocity is ZA, then motor voltage is Zero.
6. If position error is ZE and velocity is PA, then motor voltage is Zero.
7. If position error is PE and velocity is NA, then motor voltage is Positive.
8. If position error is PE and velocity is ZA, then motor voltage is Positive.
9. If position error is PE and velocity is PA, then motor voltage is Positive.

The control surface can be developed using MATLAB, as shown in Figure 4.11.

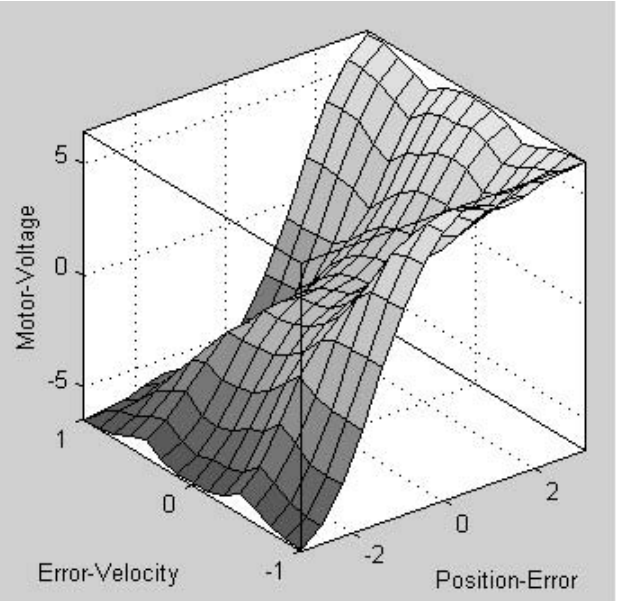


Figure 4.11. Fuzzy control surface for DC servomotor control

The system can then be simulated using the Simulink package with the system modeled in Figure 4.12. The simulation diagram presented in Figure 4.12 is slightly different from the one shown earlier in Figure 4.6. Here we have opted to use the transfer function representation of the plant rather than the state-space model, merely to illustrate various MATLAB options. MATLAB allows the conversion of state-space to transfer function, and from transfer function to state-space representation very easily. The reader is urged to experiment with these possible representations in order to build expertise in using a very powerful and useful tool.

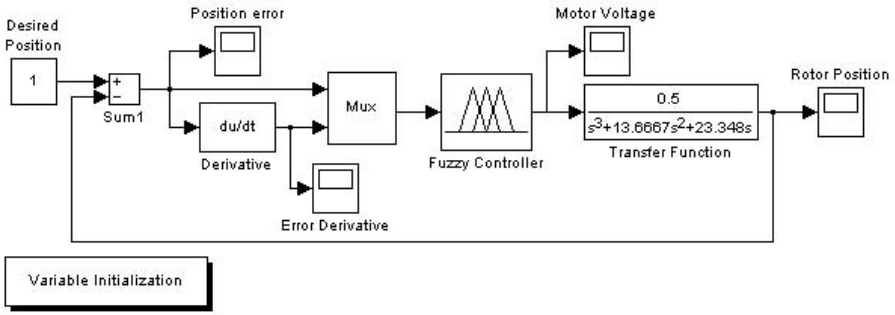


Figure 4.12. Simulink diagram for DC servomotor control using the transfer function model of the DC servomotor

Using the prototype membership functions, the scope outputs for Position Error, Error Velocity (shown as the error derivative), the actual Rotor Position, and the Fuzzy Controller output, namely the required Motor Voltage, are shown in Figure 4.13.

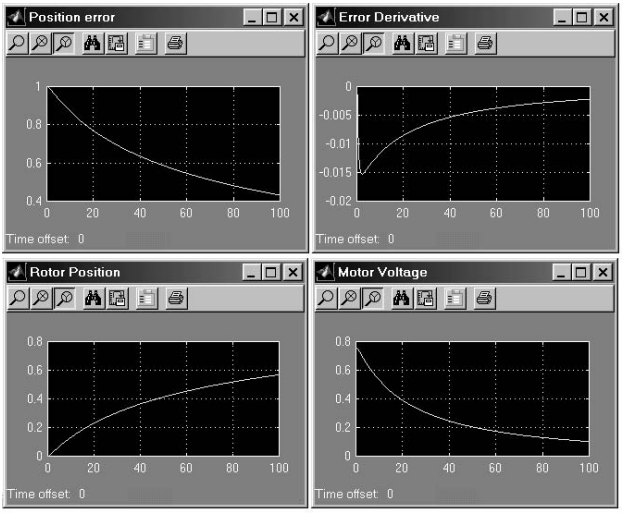


Figure 4.13. Simulated output of DC servomotor control

As in the previous example, it is clear from the simulated outputs that rapid prototyping of the controller is possible, but the membership functions have to be "tuned" to yield the desired output characteristics. We see that for a step change of 1 radian (the desired position), it takes in excess of 100 milliseconds to reach the desired value. If we focus on the membership functions for the applied voltage, we observe that for a large initial error we generate a large voltage. As the error progressively becomes small, we apply a smaller and smaller voltage, ultimately reducing the voltage to zero upon reaching the desired position. We could therefore adjust the membership function for the applied voltage so that we apply a large voltage until the rotor has reached very close to its desired

position, and then begin to decrease the voltage as the error approaches zero. There is some trial and error required here to achieve the "best" response.

As a first step, let us make the membership function labeled "ZV" in the motor-voltage set a bit narrower than the prototype chosen previously (see Figure 4.14).

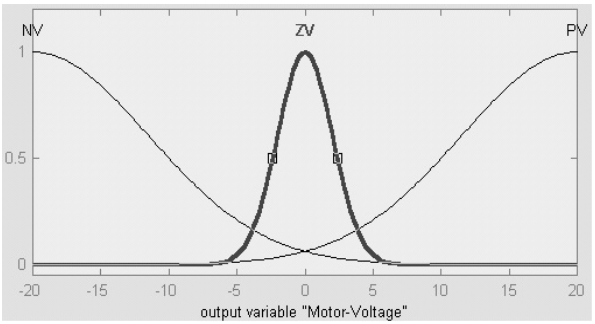


Figure 4.14. Manual adjustment of membership function

Simulating the response now yields better performance than before, with the rotor position nearly at the desired value of 1 radian in approximately 50 milliseconds. The full set of simulation results are shown in Figure 4.15.

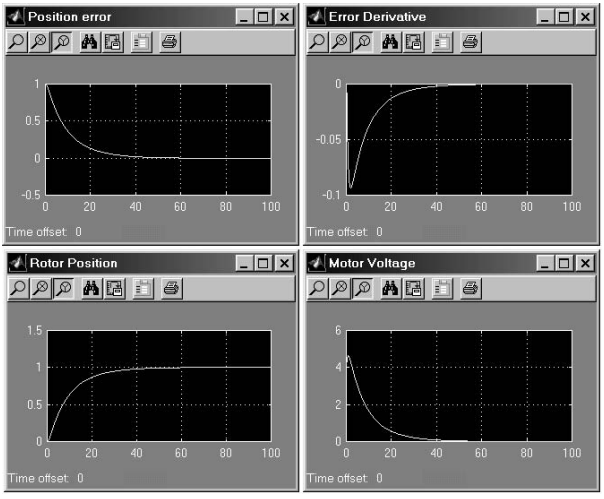


Figure 4.15. Result of adjusting membership function

Since we require a slightly faster response (a settling time of 40 milliseconds), we can make the "ZV" membership a bit narrower and try again. Suppose we reshape the membership function as in Figure 4.16 and try simulating again.

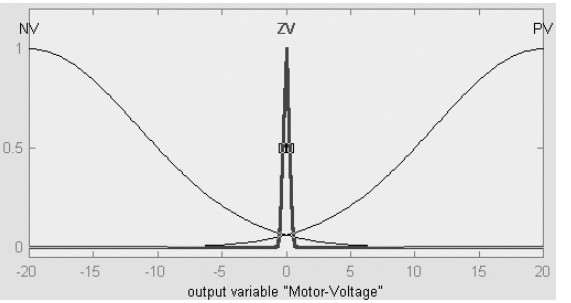


Figure 4.16. Further tuning of membership function

The simulation results in Figure 4.17 clearly show the desired response, and meet all the required specifications. We also notice that there is no overshoot. The tuning of the membership functions has succeeded. This is a much improved performance when compared to the PID controlled system.

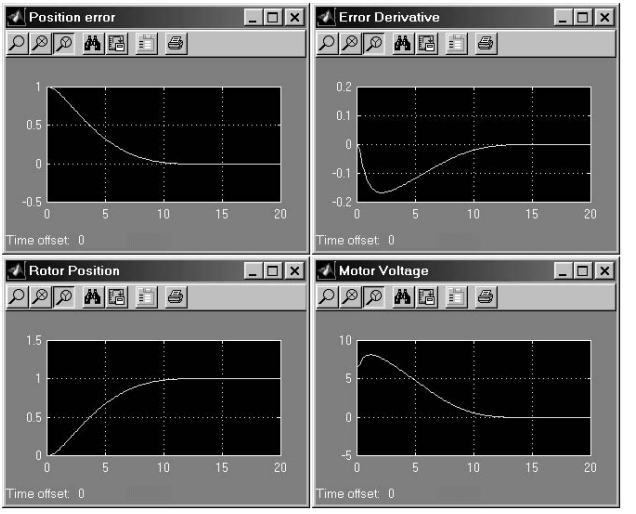


Figure 4.17. Desired simulated results

In the above set of simulations, we used a prototype fuzzy control system with three membership functions to describe each input variable. As such, our control actions were rather coarse. We used "large negative" and "large positive" values to describe the necessary control actions. While we were able to achieve the desired simulation response, we did not consider some practical issues such as power loss and the duty cycle of sensitive motor components. To take into account these issues, it would be wise to select intermediate ranges of motor voltages such as "medium negative" and "medium positive" and use these to gradually control the motor position. This would reduce the amount of fine tuning as we have seen in the example, and provide a finer granularity in the control actions. Notice that if you increase the membership functions to

five for each of the input variables and the output variable, then we have a total of 25 fuzzy rules that provide a smoother transition from one state to another. We leave this as an example for the student to develop.

4.5 Exercises and projects

1. A position control system that positions a mass m has the following open-loop transfer function

$$\frac{Y(s)}{U(s)} = \frac{40}{ms^2 + 10s + 20}$$

Using the MATLAB Simulink capabilities, address the following:

(a) For a unit step input, simulate the open-loop response of the system for $m = 1$ kg.

(b) Based on the open-loop response, it is desired to obtain a system response with settling time of less than 0.5 seconds and overshoot of less than 5%. Using a mass $m = 1$ kg, design a PID controller for the system.

(c) Using the PID parameters obtained above, simulate the performance of the system for the conditions when the mass of 1 kg is replaced by a new mass $m = 0.2$ kg and subsequently for $m = 5$ kg. Discuss the results in terms of control performance.

(d) Develop a fuzzy controller to control the position. Simulate the system behavior for various masses listed above.

(e) Compare the results from the PID controller and the fuzzy controller. What can you say about the advantages and disadvantages of using either PID or fuzzy control.

2. The open-loop transfer function of an antenna positioning system is given by

$$\frac{Y(s)}{U(s)} = \frac{5}{s^3 + 6s^2 + 5s}$$

where the output is measured in *radians*, and the input is the applied torque in *Newton-meters* produced by the drive mechanism to rotate the antenna to the desired position.

Using the MATLAB Simulink capabilities, address the following:

(a) For a unit-step input, simulate the open-loop response of the system.

(b) For any desired positioning of the antenna, it is desired that a controller produce a near critically damped system response. Design a PID controller for the system.

(c) Develop a fuzzy controller to control the antenna position.

(d) Compare the results from the PID controller and the fuzzy controller.

3. The open-loop transfer function of a plant is given by

$$G_P(s) = \frac{1}{s(s+T_1)(s+T_2)(s+T_3)}$$

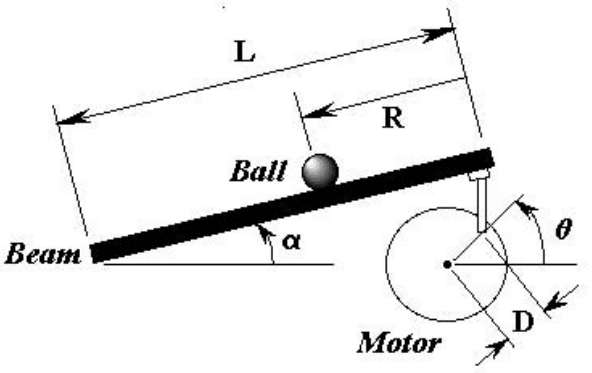
For $T_1 = 1$, $T_2 = 2$, and $T_3 = 3$, the open-loop response is unstable, while a closed-loop unity feedback system is stable. However, the response of the system is slow for a unit step input. The objective is to obtain a faster closed-loop response.

(a) Develop a Simulink model of the system and obtain a satisfactory set of PID control parameters that will improve the settling time by at least 75%. Use a less than 5% overshoot criteria.

(b) Develop a fuzzy control model that can match the performance of the PID control system.

(c) How well do the PID and fuzzy control systems perform if the parameters T_1, T_2, and T_3 vary $\pm 10\%$ from their nominal values.

4. The ball and beam problem is a nonlinear system for which we wish to examine the effectiveness of using fuzzy control. The figure below illustrates the system.



A ball of mass M placed on a beam of length L is allowed to roll along the length of the beam. A lever arm of negligible mass mounted onto a gear and driven by a servomotor is used to tilt the beam in either direction. The beam angle α is controlled by a rotational motion of the servo, shown as θ. With α initially zero, the ball is in a stationary position. When α is positive (in relation to the horizontal) the ball moves to the left due to gravity, and when α is negative the ball moves to the right. The objective in this exercise is to develop a fuzzy controller so that the ball position can be controlled to any position R along the beam.

For convenience, we derive the equation of motion for this nonlinear problem. Assuming J to represent the moment of inertia of the ball, r the

radius of the ball, and g the gravitational constant, the Lagrange equation of motion can be written as

$$\left[\frac{J}{r^2} + M\right] \ddot{R} + Mg \sin\alpha - mR\left(\dot{\alpha}\right)^2 = 0$$

The beam angle α may be approximated by a linear relationship $\alpha = \frac{D}{L}\theta$. These equations form the set of coupled equations for the system.

Use the following system parameters

M mass of the ball 0.2 kg
r radius of the ball 0.02 m
D lever arm offset 0.03 m
g gravitational acceleration 9.8 m/ s^2
L length of the beam 1.5 m
J the moment of inertia of the ball $2e^{-6}$ kg m^2

(a) Obtain a linearized model of the system for small variations in α.

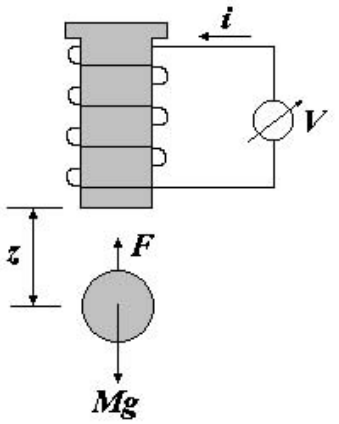
(b) For the linearized model, obtain a set of PID control parameters to satisfy a chosen criteria. Use Simulink in MATLAB for developing such a model. Test the system performance for various disturbance conditions.

(c) Using the nonlinear model, obtain a suitable fuzzy controller.

(d) Compare and contrast the performance of the PID and fuzzy controllers.

5. Magnetic levitation (MagLev) is a technology presently used in high-speed transportation systems. The idea is to levitate an object with a magnet and to propel the object at high speed. This technology has found practical use in high-speed trains in Japan and is being explored for a similar purpose in the U.S. NASA is exploring this technology for rocket propulsion as a means to reduce the fuel payload during launch.

In this exercise, the idea is to levitate a steel ball using the configuration shown in Figure 4.18. The objective is to control the vertical position z of the steel ball of mass M by controlling the current input i to the electromagnet through a variable voltage V. The force F produced by the electromagnet must overcome the force Mg created due to gravity, where g is the gravitational constant.



4.18. Magnetic levitation

The mathematical model of this system can be obtained as follows. For the steel ball,

$$M\ddot{z} = Mg - F$$

For the electrical circuit, using Kirchoff's voltage law, we can write

$$L\frac{di}{dt} + Ri = V$$

where L is the coil inductance and R is the coil resistance. The coupling equation where the force F is related to the current i is given by

$$F = k_m \frac{i^2}{z^2}$$

where k_m is the magnetic constant. Assume that the ball position is measured using an optical sensor. Use the following set of parameters in your model setup.

M	Mass of steel ball	20 milligrams (mg)
k_m	Magnetic constant	2.058 x 10^{-4} N(m/ A)^2
R	Coil resistance	0.92 Ohms (Ω)
L	Coil inductance	0.01 millihenry (m H)
i	Coil current	[0, 3] Amps (A)
V	Coil voltage	[0, 5] Volts (V)
g	Gravitational constant	9.80665 m/ s^2
z	Ball position [min, max]	[3, 7] cm

(a) Linearize the system at the minimum and maximum ball positions. For each linearization, design a PID controller using a suitable set of criteria, and simulate the performance of the system for various ball positions around the point of linearization. How well do the two separate linear models compare? Is it sufficient to obtain a single linear model that will perform equally well?

(b) Develop a fuzzy controller that can magnetically levitate the steel ball for various positions within the range indicated. Simulate the system behavior for various disturbances that can act on the steel ball.

(c) Discuss the results from the PID controller and the fuzzy controller.

6. The longitudinal motion of an aircraft is represented by a set of linear differential equations as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -0.09 & 1.0 & -0.02 \\ -8.0 & -0.06 & -6.0 \\ 0 & 0 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 10 \end{bmatrix} \delta_E$$

where,

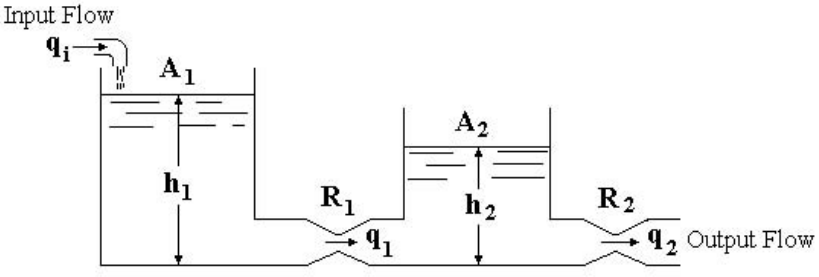
x_1 = angle of attack
x_2 = rate of change of pitch angle
x_3 = incremental elevator angle
δ_E= control input into the elevator actuator

It can be observed from the state equations that changing the elevator angle affects rate of change of pitch angle and the angle of attack.

(a) Perform open-loop simulations on the system for various inputs, namely, step, ramp, and sinusoidal inputs. What can you say about the performance of the system to such inputs?

(b) Based on the knowledge gained from open-loop simulations, develop a fuzzy controller that will maintain stable aircraft performance in the presence of unexpected disturbances.

7. A liquid level system is illustrated in the following figure. The objective in this system is to control the input flow such that desired heights of the fluid in both tanks can be maintained.



In the figure, q_i, q_1, q_2 = the rates of liquid flow, h_1, h_2 = the heights of fluid level, R_1, R_2 = the flow resistance, and A_1, A_2 = the cross-sectional tank areas.

The following basic linear relationships are valid for this system, namely,

$$\begin{aligned} q &= \tfrac{h}{R} = \text{rate of flow through orifice} \\ q_n &= (\text{tank input rate of flow}) - (\text{tank output rate of flow}) \\ &= \text{net tank rate of flow} = ADh \end{aligned}$$

Applying the above relationship to tanks 1 and 2 yields, respectively,

$$\begin{aligned} q_{n1} &= A_1 Dh_1 = q_i - q_1 = q_i - \frac{h_1 - h_2}{R_1} \\ q_{n2} &= A_2 Dh_2 = q_1 - q_2 = \frac{h_1 - h_2}{R_1} - \frac{h_2}{R_1} \end{aligned}$$

These equations can be solved simultaneously to obtain the transfer functions $\frac{h_1}{q_i}$ and $\frac{h_2}{q_i}$. The energy stored in each tank represents potential energy, which is equal to $\frac{\rho A h^2}{2}$, where ρ is the fluid density coefficient. Since these are two tanks, the system has two energy storage elements, whose energy storage variables are $h_1$and h_2. If we let $x_1 = h_1$, $x_2 = h_2$, as the state variables, and $u = q_i$ as the input to tank 1, we can write the state-variable model for the system as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{R_1 A_1} & \frac{1}{R_1 A_1} \\ \frac{1}{R_1 A_2} & -\frac{1}{R_1 A_2} - \frac{1}{R_2 A_2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{1}{A_1} \\ 0 \end{bmatrix} u$$

Letting $y_1 = x_1 = h_1$ and $y_2 = x_2 = h_2$ yields the heights of the liquid in each tank. As such, the set of output equations can be represented as

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Using MATLAB Simulink, obtain simulations that show the response of the liquid level system.

(a) Develop a set of PID control parameters that will effectively control the input so that desired heights of the fluid can be maintained in tanks 1 and 2.

(b) Develop a fuzzy control system that can effectively maintain the liquid level heights in tanks 1 and 2.

(c) Discuss the performance of the fuzzy controller and issues concerning fine tuning.

8. **Project**: A mathematical model describing the motion of a single-stage rocket is as follows:

$$\frac{d^2 y(t)}{dt^2} = c(t)\left(\frac{m}{M - mt}\right) - g\left(\frac{R}{R + y(t)}\right) - 0.5\left(\frac{dy(t)}{dt}\right)^2 \left(\frac{\rho_a A C_d}{M - mt}\right)$$

where $\frac{dy(t)}{dt}$ is the rocket velocity at time t (the plant output), $y(t)$ is the altitude of the rocket above sea level, and $c(t)$ is the plant input that controls the nozzle opening which in turn controls the velocity of the exhaust gas. The exhaust gas velocity determines the thrust of the rocket.

The following parameters are given:

- Initial mass of the rocket and fuel $M = 20,000 + 5000 = 25,000\,\text{kg}$
- Fuel consumption rate $m = 10\,\text{kg/s}$ (assumed to be constant)
- Cross-sectional area of rocket $A = 1\,\text{m}^2$
- Gravitational constant $g = 9.81456\,\text{m/s}^2$
- Radius of earth $R = 6.37 \times 10^6\,\text{m}$
- Density of air $\rho_a = 1.21\,\text{kg/m}^3$
- Drag coefficient of the rocket $C_d = 0.3$

Caution: The rocket cannot fly indefinitely!

The rocket should impact at a distance of 500 kilometers from the point of blastoff. The maximum altitude it can reach is 15 kilometers above sea level and the minimum altitude is 10 kilometers.

(a) For a selected altitude, compute the trajectory that the rocket should follow.

(b) Develop both Mamdani type and Sugeno type fuzzy controllers that control the rocket along the desired trajectory.

(c) Simulate the dynamics of motion using MATLAB or any appropriate simulation tool.

(d) Compare the performance of Mamdani and Sugeno controllers.

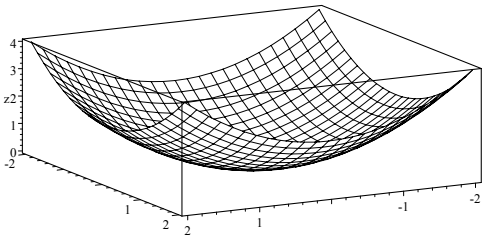
(e) Obtain a linearized model of the system and show that the system is inherently unstable.

(f) Develop an output state-variable feedback controller and simulate its performance.

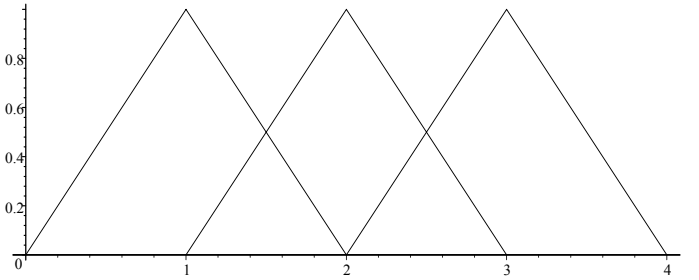
Prepare a detailed report discussing the results. Comment on the stability performance of the system due to possible disturbances.

9. **Project** (This project obtains fuzzy rules from input-output data and implements a Mamdani fuzzy inference system using these rules.)

Approximate the function $f(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$ with a Mamdani fuzzy inference system (FIS) using triangular antecedent membership functions and triangular consequent membership functions.



For the antecedent membership functions, take triangular functions on the domain $[0, 4]$.



$$A_1 = B_1 = (0, 1, 2),\ A_2 = B_2 = (1, 2, 3),\ A_3 = B_3 = (2, 3, 4)$$

Use the nine data points

If x_1 is 1 and x_2 is 1 then y is 2	If x_1 is 2 and x_2 is 3 then y is 13
If x_1 is 1 and x_2 is 2 then y is 5	If x_1 is 3 and x_2 is 1 then y is 10
If x_1 is 1 and x_2 is 3 then y is 10	If x_1 is 3 and x_2 is 2 then y is 13
If x_1 is 2 and x_2 is 1 then y is 5	If x_1 is 3 and x_2 is 3 then y is 18
If x_1 is 2 and x_2 is 2 then y is 8	

to define the antecedent functions for the nine rules

1. If x_1 is A_1 and x_2 is B_1, then y is C_1.
2. If x_1 is A_1 and x_2 is B_2, then y is C_2.
3. If x_1 is A_1 and x_2 is B_3, then y is C_3.
4. If x_1 is A_2 and x_2 is B_1, then y is C_4.
5. If x_1 is A_2 and x_2 is B_2, then y is C_5.
6. If x_1 is A_2 and x_2 is B_3, then y is C_6.
7. If x_1 is A_3 and x_2 is B_1, then y is C_7.
8. If x_1 is A_3 and x_2 is B_2, then y is C_8.
9. If x_1 is A_3 and x_2 is B_3, then y is C_9.

Use the MATLAB fuzzy toolbox to view the surface. Tune the membership functions until your surface approximates $y = x_1^2 + x_2^2$ on the domain $[1, 3] \times [1, 3]$ to your satisfaction.

Chapter 5

NEURAL NETWORKS FOR CONTROL

In this chapter, we will introduce computational devices known as neural networks that are used to solve a variety of problems, including aspects of the control of complex dynamical systems. With standard control, we rely on a mathematical model; with fuzzy control, we use a set of rules. When the information available about a system's behavior consists primarily of numerical data, the methodology of neural networks is very useful. As you will see later, some problems are best solved with a combination of fuzzy and neural approaches.

We set forth here the basic mathematical ideas used in neural networks for control. Although neural networks have applications in many fields, we will attempt to address only the parts of neural network theory that are directly applicable in control theory. We will address questions such as "What are neural networks?," "Why do we need neural networks?," and "How can we use neural networks to solve problems?" The applications of neural networks in control will be expanded upon in later chapters.

5.1 What is a neural network?

It is well known that biological systems can perform complex tasks without recourse to explicit quantitative operations. In particular, biological organisms are capable of learning gradually over time. This learning capability reflects the ability of biological neurons to learn through exposure to external stimuli and to generalize. Such properties of nervous systems make them attractive as computation models that can be designed to process complex data. For example, the learning capability of biological organisms from examples suggests possibilities for machine learning.

Neural networks, or more specifically, artificial neural networks are mathematical models inspired from our understanding of biological nervous systems. They are attractive as computation devices that can accept a large number

of inputs and learn solely from training samples. As mathematical models for biological nervous systems, artificial neural networks are useful in establishing relationships between inputs and outputs of any kind of system.

Roughly speaking, a **neural network** is a collection of artificial neurons. An **artificial neuron** is a mathematical model of a biological neuron in its simplest form. From our understanding, biological neurons are viewed as elementary units for information processing in any nervous system. Without claiming its neurobiological validity, the mathematical model of an artificial neuron [43] is based on the following theses:

1. Neurons are the elementary units in a nervous system at which information processing occurs.

2. Incoming information is in the form of signals that are passed between neurons through *connection links*.

3. Each connection link has a proper *weight* that multiplies the signal transmitted.

4. Each neuron has an internal action, depending on a *bias* or *firing threshold*, resulting in an *activation function* being applied to the weighted sum of the input signals to produce an output signal.

Thus, when input signals $x_1, x_2, \ldots, x_n$ reach the neuron through connection links with associated weights $w_1, w_2, \ldots, w_n$, respectively, the resulting input to the neuron, called the **net input**, is the weighted sum $\sum_{i=1}^{n} w_i x_i$. If the firing threshold is b and the activation function is f, then the output of that neuron is

$$y = f\left(\sum_{i=1}^{n} w_i x_i - b\right)$$

In the first computational model for artificial neurons, proposed by McCulloch and Pitts [43], outputs are binary, and the function f is the step function

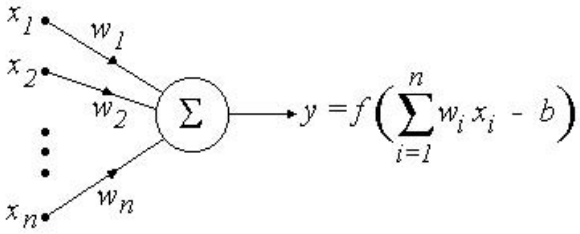


Figure 5.1. First model for artificial neuron

defined by

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

so that the activation of that neuron is

$$f\left(\sum_{i=1}^{n} w_i x_i - b\right) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq b \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i < b \end{cases}$$

This is depicted in Figure 5.1.

An artificial neuron is characterized by the parameters

$$\theta = (w_1, w_2, \ldots, w_n, b, f)$$

The bias b can be treated as another "weight" by adding an input node x_0 that always takes the input value $x_0 = +1$ and setting $w_0 = -b$ (see Figure 5.2). With this representation, adjusting bias and adjusting weights can be done in the same manner.

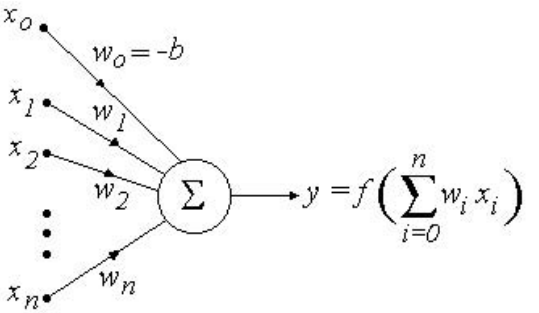


Figure 5.2. Artificial neuron with bias as weight

We will consider here only **feedforward neural networks** — that is, information propagates only forward as indicated by the direction of the arrows. Mathematically speaking, a feedforward neural network is an acyclic weighted, directed graph.

Viewing artificial neurons as elementary units for information processing, we arrive at simple neural networks by considering several neurons at a time. The neural network in Figure 5.3 consists of an **input layer** — a layer of input nodes — and one **output layer** consisting of neurons. This is referred to as a **single-layer neural network** because the input layer is not a layer of neurons, that is, no computations occur at the input nodes. This single-layer neural network is called a **perceptron**.

A **multi-layer neural network** is a neural network with more than one layer of neurons. Note that the activation functions of the different neurons can be different. The neurons from one layer have weighted connections with neurons in the next layer, but no connections between neurons of the same layer. A two-layer neural network is depicted in Figure 5.4. Note that activation functions of different neurons can be different. The input layer (or layer 0) has $n + 1$ nodes,

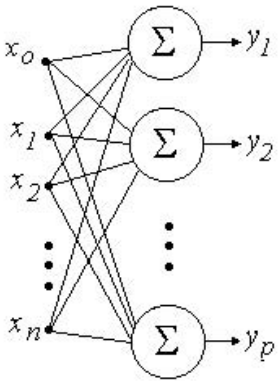


Figure 5.3. Perceptron

the middle layer, called the **hidden layer**, has p nodes, and the output layer has m nodes. This is called an "n-p-m" neural network.

Neurons (nodes) in each layer are somewhat similar. Neurons in the hidden layer are hidden in the sense that we cannot directly observe their output. From input patterns, we can only observe the output patterns from the output layer. Of course, a multi-layer neural network can have more than one hidden layer.

The two-layer neural network depicted in Figure 5.4 is a typical **multi-layer perceptron** (MLP), a multi-layer neural network whose neurons perform the same function on inputs, usually a composite of the weighted sum and a differentiable nonlinear activation function, or transfer function, such as a hyperbolic tangent function. Multi-layer perceptrons are the most commonly used neural network structures for a broad range of applications.

5.2 Implementing neural networks

Unlike computer systems that are programmed in advance to perform some specific tasks, neural networks need to be trained from examples (supervised learning) before being used. Specifically, as we will see next, a neural network can be designed and trained to perform a specific task, for example, to construct a control law for some control objective of a dynamical system.

The chosen architecture of the neural network is dictated by the problem at hand. Once the neural network architecture is chosen, we need to have training samples in order to train the neural network to perform the task intended. The training of the neural network forms the most important phase in putting neural networks to practical applications. Once the neural networks are successfully trained, they are ready to use in applications as a computational device that produces appropriate outputs from inputs.

We start out by giving an example of a logical function that can be imple-

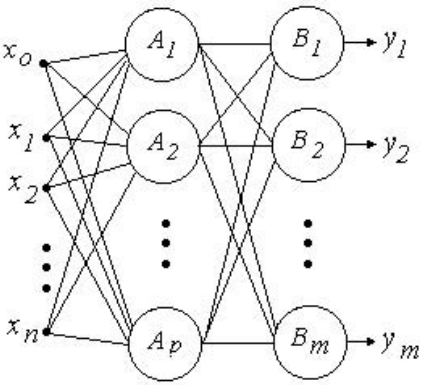


Figure 5.4. Two-layer neural network

mented by a perceptron, a neural network without hidden layers. Note that with inputs (x_1, x_2) pairs of real numbers, the domain is divided into two pieces by the activation function

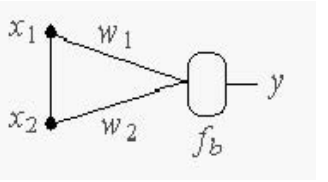
$$f(w_1x_1 + w_2x_2 - b) = \begin{cases} 1 & \text{if} \quad w_1x_1 + w_2x_2 \geq b \\ 0 & \text{if} \quad w_1x_1 + w_2x_2 < b \end{cases}$$

Namely, inputs above the line $w_1x_1 + w_2x_2 - b = 0$ produce an output of 1, and inputs below that line produce an output of 0. In particular, this setup provides a solution only if the underlying function that the neural network is trying to implement is **linearly separable** in this sense.

Example 5.1 We design a perceptron to implement the logical Boolean function OR, that is, the function $g : \{0,1\} \times \{0,1\} \to \{0,1\}$, defined by

$$g(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = x_2 = 0 \\ 1 & \text{otherwise} \end{cases}$$

In view of the function g, we consider binary-input/binary-output neural networks, with two nodes in the input layer, and only one neuron in the output layer.

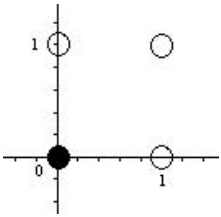


$$y = f(w_1x_1 + w_2x_2 - b)$$

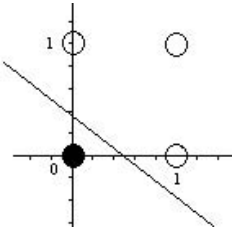
The problem is to choose w_1, w_2, and b so that

$$g(x_1, x_2) = f(w_1x_1 + w_2x_2 - b)$$

First, to see whether or not this problem is solvable, we look at the domain of the function g, coloring the points white where the function value is 1, and black where the function value is 0.



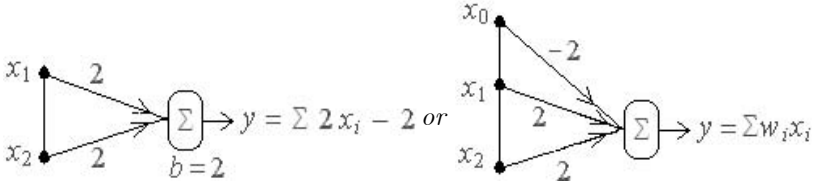
The input domain is divided into the two subsets $B = \{(0,0)\}$ and $W = \{(0,1),(1,1),(1,0)\}$ corresponding to the two output values 0 and 1, respectively. A solution to the perceptron design is a straight line $w_1x_1+w_2x_2-b=0$, in the x_1-x_2 plane, and we can see that there are lines that can separate these two subsets,



that is, this is a linearly separable problem. Of course, there are many such lines, and any such line gives a solution. In view of the simplicity of this function g, no sophisticated mathematics is needed. Just observe that

$$\begin{aligned} g(0,0) &= 0 \text{ implies } 0 < b \\ g(0,1) &= 1 \text{ implies } w_2 \geq b \\ g(1,0) &= 1 \text{ implies } w_1 \geq b \\ g(1,1) &= 1 \text{ implies } w_1 + w_2 \geq b \end{aligned}$$

and choose any numbers w_1, w_2, and b satisfying these inequalities. One solution is $w_1 = w_2 = b = 2$.



The basic logical functions are often written in set notation. For sets A and B, the set notation for the logical function AND is $A \cap B$, also called A

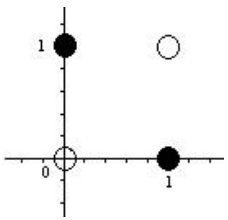
"intersection" B, and the set notation for the logical function OR is $A \cup B$, also called A "union" B.

Now consider another logical function. In the following example we consider the **exclusive OR**, which, in set notation, is written A XOR $B = (A \cap B') \cup (B \cap A')$. This set is also called the **symmetric difference** of the sets A and B, often written $A \triangle B$.

Example 5.2 The **exclusive or** (XOR) is the Boolean function with truth table

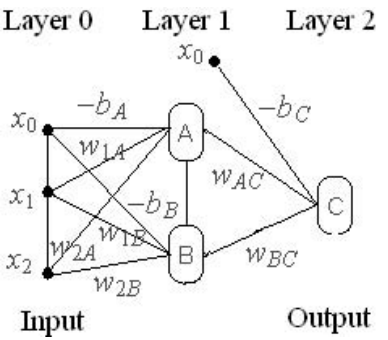
$$g(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = x_2 = 1 \\ 0 & \text{if } x_1 = x_2 = 0 \\ 1 & \text{otherwise} \end{cases}$$

When we display the input-output relationship expressed by g, we see that this problem is not linearly separable.



That is, there is no line that can separate the two subsets $W = \{(0,0),(1,1)\}$ and $B = \{(0,1),(1,0)\}$. Therefore, the function g cannot be implemented by using a perceptron, so we need to consider a hidden layer.

From Exercise 2 on page 193, you know how to implement $S \cap T'$ with weights and bias (w_{1A}, w_{2A}, b_A) and (w_{1B}, w_{2B}, b_B), respectively. From Example 5.1, we know how to implement OR with weights and bias (w_{AC}, w_{BC}, b_C). Putting these neurons together we get a solution for XOR, with the expense of having a neural network architecture more complicated than that of perceptrons, namely, using a two-layer neural network:



This is a multi-layer neural network in which the input layer (or layer 0) has two nodes x_1, x_2 (no computation), together with $x_0 = 1$ to implement the bias for neurons A and B; the hidden layer has two hidden neurons A and B, with $x_0 = 1$ to implement the bias for neuron C; and the output layer consists of

only one neuron C. For all neurons, the "activation function" f is taken to be

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Consider, for example, $x_1 = x_2 = 1$. We expect the network to produce the output $y = 0$. The output y is computed in several steps. The net input to neuron A is

$$w_{1A}x_1 + w_{2A}x_2 - b_A$$

so that the output of neuron A is $y_1 = f(w_{1A}x_1 + w_{2A}x_2 - b_A)$. Similarly, the output of neuron B is $y_2 = f(w_{1B}x_1 + w_{2B}x_2 - b_B)$. Hence, the net input to neuron C, from neurons A and B with the condition it must satisfy, is

$$w_{AC}f(w_{1A}x_1 + w_{2A}x_2 - b_A) + w_{BC}f(w_{1B}x_1 + w_{2B}x_2 - b_B) - b_C < 0$$

to result in the output $y = 0$.

The lesson is this. More layers seem to increase the computation power of neural networks. In other words, for general problems, single-layer neuron networks are not enough; multi-layer neural networks should be considered.

5.3 Learning capability

Artificial neural networks are simplified mathematical models of brain-like systems that function as parallel, distributed computing networks. As stated earlier, neural networks need to be taught or trained from examples before they can be put in use.

As we will see, the learning problem is nothing more than choosing a function from a given class of functions according to some given criteria. First, we need to know what functions can be implemented or represented by a neural network. Only after we know the answer to that, can we search for ways to design appropriate neural networks.

If we address general relationships expressed as functions from $\mathbb{R}^n$ to $\mathbb{R}^m$, then neural networks with smooth activation functions (rather than the step functions used in the early development of perceptrons for binary outputs) can approximate continuous functions with compact support. This universal approximation property of neural networks was proved using the Stone-Weierstrass theorem in the theory of approximation of functions (see Chapter 3). Specifically, all continuous functions whose domains are closed and bounded in $\mathbb{R}^n$, that is, having compact support, can be approximated to any degree of accuracy by a neural network of one hidden layer with sigmoid or hyperbolic tangent activation functions. This theoretical result means this: It is possible to design an appropriate neural network to represent any continuous function having compact support. This is only an existence theorem, not a constructive one. The significance of the theorem is that it is reassuring, since most functions of practical interest are continuous.

Then comes the basic question. How do we find an appropriate neural network to represent a given relationship? Obviously, we need some information from a given relationship in order to answer the question above. In the context of learning, the information is provided in the form of a **training set** consisting of known input-output pairs. Learning from information of this type is referred to as **supervised learning**, or learning with a teacher.

Specifically, the learning problem is this. Suppose we have a set T of training examples

$$\begin{aligned} T &= \{(x^q, y^q), q = 1, \ldots, N\} \\ x^q &= (x_1^q, x_2^q, \ldots, x_n^q) \in \mathbb{R}^n \\ y^q &= (y_1^q, y_2^q, \ldots, y_m^q) \in \mathbb{R}^m \end{aligned}$$

and we wish to use this data to adjust the weights and biases of a neural network with n input nodes, and m output neurons with one hidden layer, for example. Then from a common-sense viewpoint, we should compare the output

$$o^q = (o_1^q, o_2^q, \ldots, o_m^q) \in \mathbb{R}^m$$

from the input pattern x^q, with the corresponding target output y^q. This is the same as approximating a function from which the input-output pairs have been drawn.

The error correction idea is simple: A change in weights should be made or not according to the comparison of the actual output with the desired output. This idea is formulated in terms of a suitable overall performance measure. In a general sense, the learning problem may be stated as follows: Given a class of functions (here an architecture of neural networks) and a performance criterion, find a function in this class that optimizes the performance criterion on the basis of a set of training examples.

In a sense, neural networks are a class of learning machines, and the back-propagation algorithm is a special case of a general inductive principle called the empirical risk minimization principle [76]. As such, the learning capability of neural networks falls under the same line of analysis as learning machines. We indicate here the ideas used in the analysis.

Recall that a control law is a function φ from an input space X to an output space Y that predicts the output for a given input. If we do not have a mathematical model, or even if we do but do not have an analytic method to find φ from it, we need to look for other alternatives.

Consider the situation where the information that will help us find φ is a set of desirable pairs (x_i, y_i), $i = 1, \ldots, m$, where $y_i = \varphi(x_i)$. Then the obvious problem is to find a function φ_N that fits through these points in such a way that prediction is good — that is, for new x the fitted function φ_N will produce values $\varphi_N(x)$ close enough to $\varphi(x)$.

To carry out the program above, we need the specification of performance criteria and a way to construct φ_N that meets these criteria. The performance criterion for a good approximation is specified as an acceptable error of approximation. A neural network is a tool that can be used to achieve these two

requirements simultaneously. However, as with all tools, neural networks have limitations.

The neural network approach consists of two phases — creating a set of models and providing a mechanism for finding a good approximation. First, the architecture of a neural network provides a set $\mathcal{F}$ of functions from X to Y. The fact that $\mathcal{F}$ is a good set of models for φ is due to the **universal approximation property** of neural networks — that is, φ can be approximated by some element in $\mathcal{F}$. Then the computer-oriented **backpropagation algorithms** of multilayer neural networks are designed to implement this approximation.

Instead of having the desired pairs, or sample data, suppose we only have experts' knowledge as information to find φ. Then we are in the domain of applicability of fuzzy control. The set of models is created by experts' linguistic "If...then..." rules. A similar universal approximation property of fuzzy systems exists. However, the search for a good approximation is done by tuning — that is, there is no systematic procedure to reach the approximate function φ_N. This drawback can be compensated by using neural networks in fuzzy control, leading to what is called **neural-fuzzy control**, in which the formation used to construct φ is a fuzzy rule base represented in a neural network structure so that the systematic search by backpropagation algorithms of neural networks can be used to find a good approximation for φ. Of course, this requires experiments on the controlled systems to extract samples. The advantage to this approach is that we have incorporated all available information (experts' knowledge and numerical data) in our process of constructing the control law φ. This process replaces the tuning process and is referred to as **optimization of fuzzy rules**. Indeed, when linguistic labels in fuzzy rules are modeled by parametric membership functions, the learning algorithm of neural networks will optimize the parameters in fuzzy rules, leading to the desired approximation of φ.

With the above program in mind, it is necessary to assess the learning capability of neural networks. An excellent study of this issue can be found in [4]. Roughly speaking, although the universal approximation property states that neural networks can approximate a large class of functions, in practice we also need to know how much training data is needed to obtain a good approximation. This clearly depends on the complexity of the neural network considered.

To give a measure of this complexity, we will define the Vapnik-Chervonenkis dimension of the class of functions computable by the neural network. Consider the perceptron with n real inputs and binary outputs $\{0,1\}$. Suppose we have a training sample $T = \{(x_i, y_i)\,,\ i = 1, \ldots, n\}$ with $x_i \in \mathbb{R}^n$ and $y_i \in \{0,1\}$. Let $S = \{x_1, x_2, \ldots, x_m\} \subset \mathbb{R}^n$ with $|S| = m$, and let $\mathcal{F}$ be the class of all functions computable (representable) by the perceptron. Here, $\mathcal{F}$ can be thought of as a class of subsets of $\mathbb{R}^n$, since each element of $\mathcal{F}$ is a function from $\mathbb{R}^n$ to $\{0,1\}$. Let $\mathcal{F}_S$ be the class of restrictions of functions in $\mathcal{F}$ to S, and identify the class $\mathcal{F}_S$ with a collection of subsets of S. If $|\mathcal{F}_S| = 2^{|S|} = 2^m$, we say that $\mathcal{F}$ **shatters** S. The **growth function** of the class $\mathcal{F}$ is a special map

$$G_{\mathcal{F}} : N \to N$$

where $N = \{0, 1, 2, ...\}$ is the set of natural numbers. The growth function is

defined by

$$G_{\mathcal{F}}(k) = \max\{|\mathcal{F}_A| : A \subset \mathbb{R}^n,\ |\mathcal{A}| = k\}$$

Note that for all $k \in N$,

$$G_{\mathcal{F}}(k) \leq 2^k$$

The **Vapnik-Chervonenkis dimension** (VC dimension) of $\mathcal{F}$ is the size of the largest shattered finite subset of $\mathbb{R}^n$, or equivalently, the largest value of k for which $G_{\mathcal{F}}(k) = 2^k$.

In the case of the perceptron, we have

$$\begin{aligned} G_{\mathcal{F}}(k) &= 2\sum_{k=0}^{n}\binom{m-1}{k} \\ &= \begin{cases} 2^m & \text{for } n \geq m-1 \\ 2^m - 2\sum_{k=n+1}^{m-1}\binom{m-1}{k} & \text{for } n < m-1 \end{cases} \end{aligned}$$

so that the VC dimension of $\mathcal{F}$, denoted by $D(\mathcal{F})$, is $n+1$.

For general neural networks with real-valued outputs, the class of $\mathcal{F}$ of computable functions is a class of functions. The concept of VC dimension of a class of sets can be extended to the case of a class of functions by using subgraphs. The **subgraph** of a function f is the subset of $\mathbb{R}^n \times \mathbb{R}$ defined by

$$S(f) = \{(x,t) : x \in \mathbb{R}^n,\ t \in \mathbb{R},\ t \leq f(x)\}$$

Then $D(F)$ is defined as the VC dimension of the class of its subgraphs

$$\{S(f) : f \in F\}$$

Essentially, neural networks with finite VC dimensions are trainable.

5.4 The delta rule

The delta rule is a learning algorithm for *single-layer neural networks*. We choose to present this learning algorithm [81] in some detail since it is a precursor of the backpropagation algorithm for multi-layer neural networks.

The idea is to define a measure of the overall performance of a network, such as the one shown in Figure 5.5, then find a way to optimize that performance. Obviously, learning algorithms should change the weights so that output o^q becomes more and more similar to the target output y^q for all $q = 1, 2, \ldots, m$, when presenting the input x^q to the network.

A suitable overall performance measure is

$$E = \sum_{q=1}^{N} E^q$$

where

$$E^q = \frac{1}{2}\sum_{i=1}^{m}(y_i^q - o_i^q)^2$$

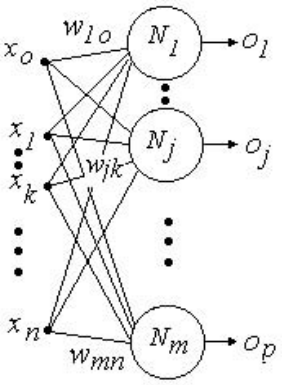


Figure 5.5. Network with weights w_{ij}

The goal is to minimize E with respect to the weights w_{ij} of the network.

Let w_{ij} denote the weight from the node j to the output neuron i. To use the *gradient descent method* for optimization, we need the w_{ij}'s to be differentiable. This boils down to requiring

$$o_i^q = f_i\left(\sum_{j=0}^{n} w_{ij} x_j^q\right)$$

to be differentiable. That is, the activation function f_i of the i^{th} neuron should be chosen to be a differentiable function. Note that step functions such as

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

are not differentiable.

The sigmoid activation function, shown in Figure 5.6, is differentiable. Note

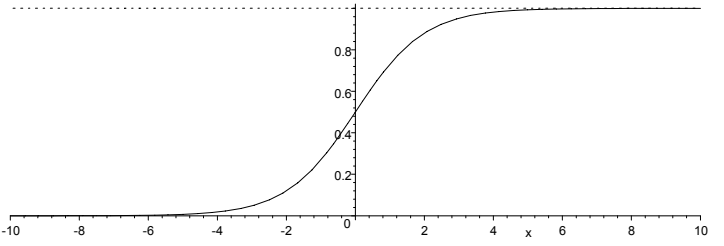


Figure 5.6. $f(x) = \frac{1}{1+e^{-x}}$

that a step activation function models neurons that either fire or do not fire, while continuous activation functions model neurons such as sigmoids that provide a gradual degree of firing — a "fuzzy property" that is more realistic.

The error E is a function of the variables w_{ij}, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$. Recall that the **gradient** ∇E of E at a point w with components w_{ij} is the vector of partial derivatives $\frac{\partial E}{\partial w_{ij}}$. Like the derivative of a function of one variable, the gradient always points to the uphill direction of the function E. The downhill (steepest descent) direction of E at W is $-\nabla E$. Thus, to minimize E, we move proportionally to the negative of ∇E, leading to the updating of each weight w_{jk} as

$$w_{jk} \longrightarrow w_{jk} + \triangle w_{jk}$$

where

$$\triangle w_{jk} = -\eta \frac{\partial E}{\partial w_{ij}}$$

and $\eta > 0$ is a number called the **learning rate**.

We have

$$\frac{\partial E}{\partial w_{ij}} = \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{ij}}$$

and

$$\begin{aligned} \frac{\partial E^q}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \left(\frac{1}{2} \sum_{i=1}^{m} \left(y_i^q - o_i^q \right)^2 \right) \\ &= \left(o_j^q - y_j^q \right) \frac{\partial}{\partial w_{ij}} f_j \left(\sum_{i=0}^{n} w_{ji} x_i^q \right) \end{aligned}$$

since

$$\frac{\partial}{\partial w_{ij}} \left(o_i^q - y_i^q \right) = 0 \text{ for } i \neq j$$

noting that

$$o_j^q = f_j \left(\sum_{i=0}^{n} w_{ji} x_i^q \right)$$

Thus,

$$\begin{aligned} \frac{\partial E^q}{\partial w_{jk}} &= x_k^q \left(o_j^q - y_j^q \right) f_j' \left(\sum_{i=0}^{n} w_{ji} x_i^q \right) \\ &= \delta_j^q \cdot x_k^q \end{aligned}$$

where

$$\delta_j^q = \left(o_j^q - y_j^q \right) f_j' \left(\sum_{i=0}^{n} w_{ji} x_i^q \right)$$

for the j^{th} output neuron.

Therefore,

$$\begin{aligned} \triangle w_{jk} &= \sum_{q=1}^{N} \triangle^q w_{jk} \\ &= -\eta \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{jk}} \\ &= \sum_{q=1}^{N} -\eta\, \delta_j^q x_k^q \end{aligned}$$

The delta rule leads to the search for a weight vector w^* such that the gradient of E at w^* is zero. For some simple single-layer neural networks, w^* is the unique absolute minimum of $E(w)$.

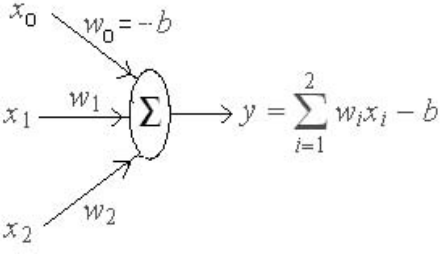
Example 5.3 In this example, we implement the logical function AND by minimizing the error E. Consider the training set T consisting of binary-inputs/bipolar-targets:

$$T = \{(x^q, y^q),\ q = 1, 2, 3, 4\}$$

with $x^q \in \{0,1\}^2$ and $y^q \in \{-1, 1\}$. Specifically,

$$\begin{array}{ll} x^1 = \left(x_1^1, x_2^1\right) = (1,1) & y^1 = 1 \\ x^2 = \left(x_1^2, x_2^2\right) = (1,0) & y^2 = -1 \\ x^3 = \left(x_1^3, x_2^3\right) = (0,1) & y^3 = -1 \\ x^4 = \left(x_1^4, x_2^4\right) = (0,0) & y^4 = -1 \end{array}$$

An appropriate neural network architecture for this problem is the following, with linear activation function $f(x) = x$.



We have

$$E = \sum_{q=1}^{4} \left(x_1^q w_1 + x_2^q w_2 - w_0 - y^i\right)^2$$

The weight vector (w_0^*, w_1^*, w_2^*) that minimizes E is the solution of the system of equations

$$\frac{\partial E}{\partial w_j} = 0,\ j = 0, 1, 2$$

Explicitly,

$$\begin{aligned}
(1) \quad \frac{\partial E}{\partial w_0} &= w_1 + w_2 - 2w_0 + 1 = 0 \\
(2) \quad \frac{\partial E}{\partial w_1} &= 2w_1 + w_2 - 2w_0 = 0 \\
(3) \quad \frac{\partial E}{\partial w_2} &= w_1 + 2w_2 - 2w_0 = 0
\end{aligned}$$

Here, the solution of this system of linear equations is easy to obtain. Indeed, subtracting (3) from (1) yields $w_2 = 1$. With $w_2 = 1$, (2) and (3) yield

$$2w_0 = 2w_1 + 1 = w_1 + 2$$

which implies $w_1 = 1$. Then from (3), we get

$$w_0 = \frac{w_1 + 2w_2}{2} = \frac{3}{2}$$

Note that in the delta rule, as well as in the generalized delta rule that we will consider next, we need to calculate the derivatives of the activation functions involved. The choice of smooth activation functions is dictated by the ranges of output variables. For example, if the range of the output of a neuron, for some specific application, is the open interval $(0, 1)$, then an appropriate differentiable activation function could be the sigmoid function shown in Figure 5.7. If the output variable has range $(-1, 1)$, then an appropriate activation

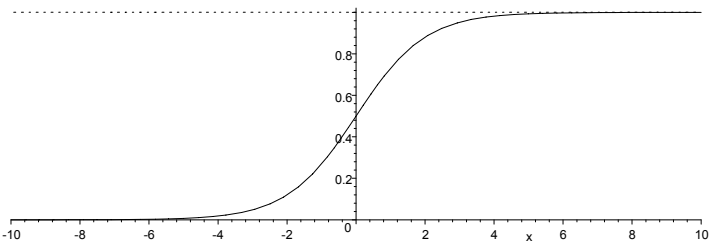


Figure 5.7. $f(x) = \frac{1}{1+e^{-x}}$

function should have the same range, for example $f(x) = \frac{2}{1+e^{-x}} - 1$.

5.5 The backpropagation algorithm

As we have seen from a simple example in the implementation of the exclusive or Boolean function XOR, we need to consider multi-layer neural networks for approximating general relationships. So we need learning algorithms for these more complicated neural networks.

The following popular learning algorithm, referred to as the **backpropagation algorithm**, is a generalization of the delta rule [42]. If we look closely at the delta rule for single-layer neural networks, we realize that to update a weight w_{ik} when the learning input pattern x^q is presented, we need

$$\triangle w_{ik}^q = -\eta\, \delta_i^q x_k^q$$

That is, we need to compute the function

$$\delta_i^q = (o_i^q - y_i^q)\, f'\left(\sum_{j=0}^{n} w_{ij} x_j^q\right)$$

and this is possible since we have at our disposal the value y_i^q that is known to us as the target output for the output node i.

Consider, for simplicity, the two-layer n-m-p neural network depicted in Figure 5.8. It seems that we cannot update a weight like w_{ik} on the link connecting

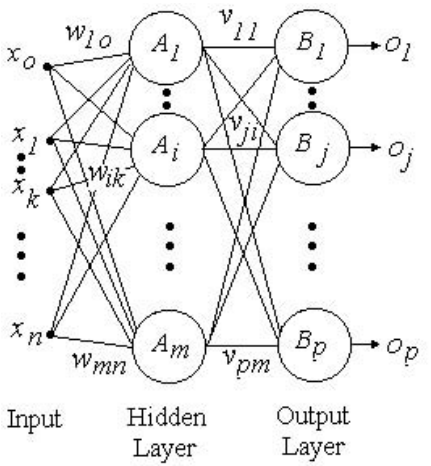


Figure 5.8. Two-layer neural network

the k^{th} input node in the input layer to the i^{th} hidden neuron in the hidden layer, since we are not given the target pattern of the i^{th} neuron from the input x^q. Note that patterns y^q are target patterns of neurons in the output layer, and not the hidden layers. Thus, when we look at the errors $\left(o_j^q - y_j^q\right)^2$ at the output layer, we cannot detect which hidden neurons are responsible.

It turns out that to update these weights, it suffices to be able to compute $\frac{\partial E}{\partial o_i}$, the partial derivative of the global error E with respect to the output o_i, $i = 1, \ldots, p$. For differentiable activation functions, the gradient descent strategy can still be applied to find the network's weight configuration w^* that minimizes the error E.

We will write down the updating formulas for weights in a two-layer neural network. The generalization to more than two layers is just a matter of notation. Let v_{ji} be the weight of the link connecting the hidden neuron i to the output neuron j, and w_{ik} the weight of the link connecting the input node k to the hidden neuron i.

First, the updating rule for the v_{ji}'s is the same as in the delta rule, by viewing now the hidden layer as an input layer. For output neuron j and hidden neuron i, the weight v_{ji} is updated using

$$\begin{aligned} \triangle v_{ji} &= \sum_{j=1}^{N} \triangle^q v_{ji} \\ &= -\eta \sum_{j=1}^{N} \frac{\partial E^q}{\partial v_{ji}} \\ &= \sum_{j=1}^{N} \left(-\eta\, \delta_j^q z_i^q\right) \end{aligned}$$

where z_i^q is the net input to the hidden neuron i,

$$z_i^q = f_i \left(\sum_{k=0}^{n} w_{ik} x_k^q \right)$$

and

$$\delta_j^q = \left(o_j^q - y_j^q\right) f_i' \left(\sum_{k=1}^{m} v_{jk} z_k^q \right)$$

For a hidden neuron i and input node k, the weight w_{ik} is updated as follows:

$$\triangle^q w_{ik} = -\eta \frac{\partial E^q}{\partial w_{ik}}$$

with

$$\frac{\partial E^q}{\partial w_{ik}} = \frac{\partial E^q}{\partial o_i^q} \frac{\partial o_i^q}{\partial w_{ik}}$$

where o_i^q is the output of the hidden neuron i,

$$o_i^q = f_i \left(\sum_{\ell=0}^{n} w_{i\ell} x_\ell^q \right) = z_i^q$$

We have

$$\frac{\partial o_i^q}{\partial w_{ik}} = f_i' \left(\sum_{\ell=0}^{n} w_{i\ell} x_\ell^q \right) x_k^q$$

Let $\delta_i^q = \frac{\partial E^q}{\partial o_i^q}$, the hidden layer. How do we compute this? Observe that

$$\delta_i^q = \frac{\partial E^q}{\partial o_i^q} = \sum_{j=1}^{p} \frac{\partial E^q}{\partial o_j^q} \frac{\partial o_j^q}{\partial o_i^q}$$

where j is in the output layer. The $\frac{\partial E^q}{\partial o_j^q}$ are known from previous calculations using the delta rule. Next,

$$o_i^q = f_j' \left(\sum_{\ell=1}^{m} v_{i\ell} z_\ell^q \right) v_{ji}$$

Thus the δ_i^q, for hidden neurons i, are computed from the already-known values of δ_j^q for all j in the output layer.

Because of this fact, this generalized delta rule is called the backpropagation algorithm. First feed the input patterns x^q forward to reach the output layer and then calculate the δ_j^q's for all output neurons j. Next propagate these δ_j^q's backward to the layer below (here, the hidden layer) in order to calculate the δ_i^q's for all neurons i of that layer.

There are two strategies for training.

1. **The batch approach**: The weights w_{ik} are changed according to

$$\triangle w_{ik} = -\eta \sum_{q=1}^{N} \frac{\partial E^q}{\partial w_{ik}}$$

   after all N training patterns are presented to the neural network.

2. **The incremental approach**: Change the weights w_{ik} after every training pattern q is presented to the neural network, that is, using

$$\triangle^q w_{ik} = -\eta \frac{\partial E^q}{\partial w_{ik}}$$

   to update w_{ik}.

Note that the batch update $\triangle w_{ik}$ is just the sum of the incremental updates $\triangle^q w_{ik}$, $q = 1, 2, \ldots, N$.

In the backpropagation algorithm, we start out by the calculation of the δ value in the output layer. Then we propagate the error vector backward from the output layer towards the input terminals. When all the weights are updated with $\triangle^q w_{ik}$, the next training pattern x^q is presented, and the procedure starts again. The **stopping criterion** can be either a threshold or the error function E.

In summary, the backpropagation algorithm can be described by the following steps.

1. Initialize weights w_{ik} with small random values, and select the learning rate η.

2. Apply the pattern x^q to the input layer.

3. Propagate x^q forward from the input layer to the output layer using

$$o_i = f_i\left(\sum_{k=0}^{p} w_{ik} o_k\right)$$

4. Calculate the error $E^q(w)$ on the output layer using

$$E^q(w) = \frac{1}{2}\sum_{i=1}^{p}\left(o_i^q - y_i^q\right)^2$$

5. Calculate the δ values of the output layer using

$$\delta_i^q = f_i'\left(\sum_{k=1}^{n} v_{ik} z_k\right)\left(o_i^q - y_i^q\right)$$

6. Calculate the δ values of the hidden layer by propagating the δ values backward, that is,

$$\delta_i^q = f_i'\left(\sum_{k=0}^{n} w_{ik} x_k^q\right)\sum_{j=1}^{p} v_{ij}\delta_j^q$$

7. Use

$$\triangle^q w_{ik} = -\eta\,\delta_i^q o_k^q$$

for all w_{ik} of the neural network.

8. $q \longrightarrow q+1$ and go to step 2.

Both batch and incremental approaches can be applied for updating the weight configuration w. The relative effectiveness of the two approaches depends on the problem, but the incremental approach seems superior in most cases, especially for very regular or redundant training sets.

The algorithm leads to w^*, such that

$$\triangle(E)(w^*) = 0$$

which could be a local minimum.

5.6 Example 1: training a neural network

We want to train the two-layer perceptron network shown in Figure 5.9 to respond with a desired output $d = 0.9$ at the output y_1 to the augmented input vector $x = [1, x_1, x_2]^T = [1, 1, 3]^T$. The network weights have been initialized as shown. Assuming sigmoidal activation functions at the outputs of the hidden and output nodes and learning gains of $\eta = 0.1$ and no momentum term, we

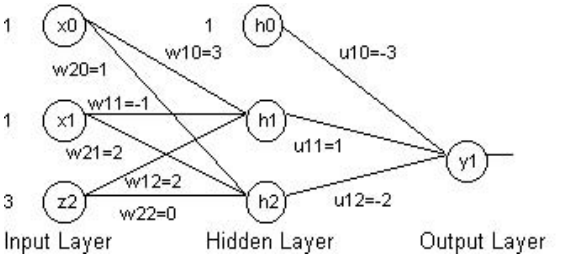


Figure 5.9. Two-layer feedforward perceptron

analyze a single feedforward and backpropagation step for the initialized network with activation function $f(z) = \frac{1}{1+e^{-z}}$.
The weight matrices for the specified network are

$$\begin{aligned} d &= 0.9 \\ x &= \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} \\ W &= \begin{bmatrix} w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 0 \end{bmatrix} \\ u &= \begin{bmatrix} u_{10} \\ u_{11} \\ u_{12} \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \\ -2 \end{bmatrix} \end{aligned}$$

The output of the hidden node is given by $a_i = w_j \bullet x$ and $h_j = f(a_j) = \frac{1}{1+e^{-a_j}}$, from which we calculate

$$\begin{aligned} a_1 &= (3)(1) + (-1)(1) + (2)(3) = 8 \\ a_2 &= (1)(1) + (2)(1) + (0)(3) = 3 \\ h &= [h_0, h_1, h_2]^T = [1, 0.9997, 0.9526]^T \end{aligned}$$

The network output is therefore

$$\begin{aligned} y_1 &= f(u \cdot h) = f((-3)(1) + (1)(0.9997) - (2)(0.9526)) \\ &= f(-3.9055) = 0.0197 \end{aligned}$$

The error signals are computed as follows:

$$\begin{aligned} \delta y_1 &= (y_1 - d)(y_1)(1 - y_1) \\ &= (0.0197 - 0.9)(0.0197)(1 - 0.0197) = -0.0170 \\ \delta h_1 &= (h_1)(1 - h_1)(\delta y_1)(u_{11}) \\ &= (0.9997)(1 - 0.9997)(-0.0197)(1) = -5.098 \times 10^{-6} \\ \delta h_2 &= (h_2)(1 - h_2)(\delta y_1)(u_{12}) \\ &= (0.9526)(1 - 0.9526)(-0.0170)(-2) = 8.056 \times 10^{-4} \end{aligned}$$

Compute all the nine weight updates:

1. $\Delta w_{10} = -\eta(\delta h_1)(x_0) = -(0.1)(-5.098 \times 10^{-6})(1) = 5.098 \times 10^{-7}$
2. $\Delta w_{11} = -\eta(\delta h_1)(x_1) = -(0.1)(-5.098 \times 10^{-6})(1) = 5.098 \times 10^{-7}$
3. $\Delta w_{12} = -\eta(\delta h_1)(x_2) = -(0.1)(-5.098 \times 10^{-6})(3) = 1.5294 \times 10^{-6}$
4. $\Delta w_{20} = -\eta(\delta h_2)(x_0) = -(0.1)(8.056 \times 10^{-4})(1) = -8.056 \times 10^{-5}$
5. $\Delta w_{21} = -\eta(\delta h_2)(x_1) = -(0.1)(8.056 \times 10^{-4})(1) = -8.056 \times 10^{-5}$
6. $\Delta w_{22} = -\eta(\delta h_2)(x_2) = -(0.1)(8.056 \times 10^{-4})(3) = -2.417 \times 10^{-4}$
7. $\Delta u_{10} = -\eta(\delta y_1)(h_0) = -(0.1)(-0.0170)(1) = 0.0017$
8. $\Delta u_{11} = -\eta(\delta y_1)(h_1) = -(0.1)(-0.0170)(0.9997) = 0.0017$
9. $\Delta u_{12} = -\eta(\delta y_1)(h_2) = -(0.1)(-0.0170)(0.9526) = 0.0016$

Using the delta rule, the new weights are updated as

$$W = \begin{bmatrix} w_{10} + \Delta w_{10} & w_{11} + \Delta w_{11} & w_{12} + \Delta w_{12} \\ w_{20} + \Delta w_{20} & w_{21} + \Delta w_{21} & w_{22} + \Delta w_{22} \end{bmatrix}$$

$$u = \begin{bmatrix} u_{10} + \Delta u_{10} \\ u_{11} + \Delta u_{11} \\ u_{12} + \Delta u_{12} \end{bmatrix}$$

In this example, we see that since the weight updates are very small, the network is very nearly trained to obtain the desired output $d = 0.9$. Repeated computations will make the output y_1 tend toward any desired near-zero tolerance.

5.7 Example 2: training a neural network

In this example, we discuss a practical application of significance to the cotton ginning industry. Typically, when cotton undergoes ginning, trash particles from the harvesting process have to be removed. Trash is comprised of bark from the cotton plants, pieces of stems (sticks), leaves, crushed leaves (pepper), and other extraneous matter from the cotton farms. The ginning process comprises machines that can remove most or all of the trash particles. The final grade of ginned cotton is determined by the amount of remaining trash that cannot be removed from the cotton without seriously affecting the material properties. Identifying trash, therefore, constitutes a significant step in optimizing the number of trash-removing machines that can be placed on line during the ginning process.

Identifying trash particles requires that we identify the features of each type of trash, and determine which features can be used to distinguish between "bark," "stick," "leaf," and "pepper." Once we have the features, we can choose the appropriate neural network architecture, and train the neural network to

perform trash identification. We explain here how a backpropagation neural network was trained to identify four different trash types in cotton, namely, bark, stick, leaf, and pepper [65]. The training data used to train the neural network consisted of a large number of samples for which image processing techniques were used to determine the features of each trash type. Different neural network topologies were trained and tested to evaluate their classification performance. Results from two neural network topologies — one with a single hidden layer and an output layer and the other with two hidden layers and an output layer — are presented here for comparative purposes.

The fundamental objective in object recognition is to utilize a minimum number of features that can be used not only to identify a specific object, but also to distinguish between objects. To accomplish this objective, we need to obtain the right combination of distinguishing features. Extracting features of objects that do not have any mathematical representation is based upon empirical relationships that provide some measure of the object shape. Objects such as "bark," "stick," "leaf," and "pepper" have characteristics that must be obtained from image analysis of such objects. Figures 5.10–5.13 illustrate samples of trash objects and their corresponding computer-segmented images that were used in obtaining features. Numbers alongside the segmented objects are object identifiers that are used to tag the computed features to that object.

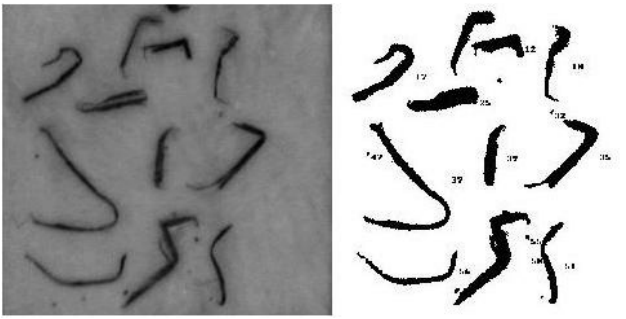


Figure 5.10. Bark sample and segmented image

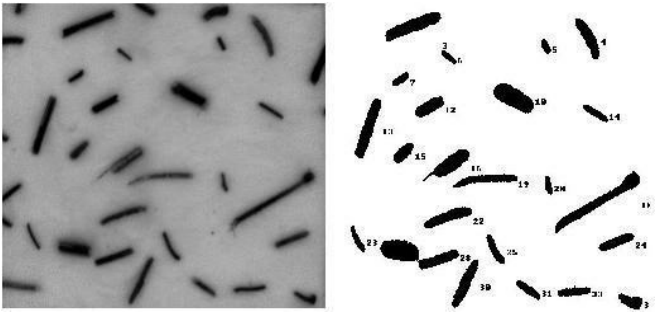


Figure 5.11. Stick sample and segmented image

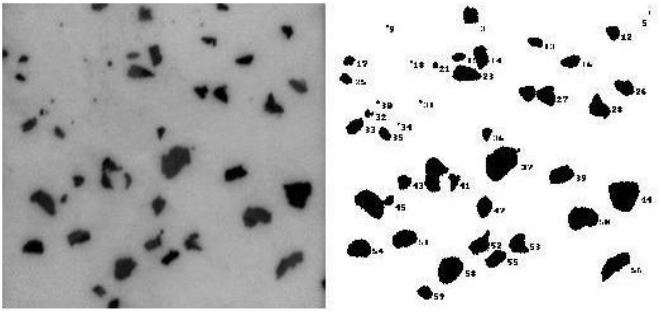


Figure 5.12. Leaf sample and segmented image

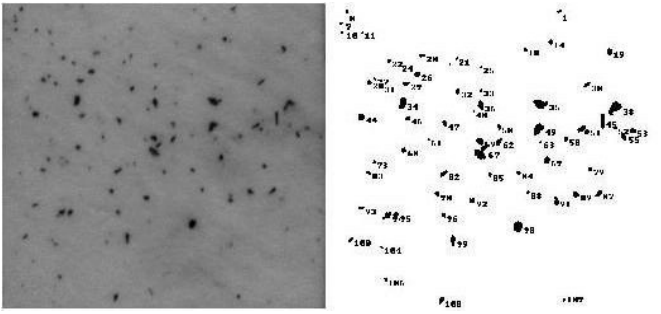


Figure 5.13. Pepper sample and segmented image

Table 5.1 lists two sets of features that were selected to train backpropagation neural networks with various topologies. The purpose of course was to determine which set of features yielded the best recognition using an appropriate network topology.

Table 5.1. Features used to train neural network

Feature set 1	Feature set 2
Area, Solidity, and E^{dif}	Area, Shape Factor, Solidity, Convexity, Compactness, Aspect Ratio, and E^{dif}

The feature parameters listed in Table 5.1 are described by empirical relationships. They are primarily shape descriptors and are briefly described here for completeness. The methods for obtaining these features are well documented and are available in the MATLAB image processing toolbox. A list of several other descriptors can be obtained from any image processing handbook.

Area: Since images are represented in terms of pixels, the area of an object is the total number of pixels within the object.

Perimeter: The perimeter is the number of pixels along the boundary of the object.

Shape Factor: This feature is defined as $\frac{\text{Perimeter}^2}{4\pi \cdot \text{Area}}$.

Convex Area: The computation of this feature is somewhat involved. The best measure for this feature is obtained by constructing a many-sided polygon around the object and counting the total number of pixels within the bounding polygon.

Solidity: This is defined as $\frac{\text{Area}}{\text{Convex Area}}$.

Convex Perimeter: This feature is computed by bounding an object with a many-sided irregular polygon and counting the total number of pixels along the perimeter.

Convexity: This is defined as $\frac{\text{Convex Perimeter}}{\text{Perimeter}}$.

Maximum and Minimum Diameter: These parameters are obtained by computing the maximum and minimum number of pixels across a many-sided polygon bounding an object.

Aspect Ratio: This is defined as $\frac{\text{Maximum Diameter}}{\text{Minimum Diameter}}$.

Compactness: This feature is defined as $\frac{\sqrt{\left(\frac{4}{\pi}\right) \cdot \text{Area}}}{\text{Maximum Diameter}}$.

Bounding Box Area: This feature is obtained by determining the area of a rectangle circumscribing the object with sides of the rectangle parallel to the image edges.

Extent: This is defined as $\frac{\text{Area}}{\text{Bounding Box Area}}$.

E^{dif}: This is the variation in *Extent* when an object is rotated by $\pm\frac{\pi}{4}$ radians.

Using the shape descriptors, a set of training patterns and a set of test patterns were generated for each trash type. For training the neural network, the targets for bark, stick, leaf, and pepper were arbitrarily chosen as 0.2, 0.4, 0.6, and 0.8, respectively. Several neural network topologies were examined. In all cases, the activation function chosen was the unipolar sigmoidal function. The neural networks were trained to classify trash objects as follows: if the output is between 0.2 ± 0.1, then the trash object is classified as "bark." Similarly, the object is classified as "stick" if the output is between 0.4 ± 0.1, "leaf" if the output is between 0.6 ± 0.1, and "pepper" if the output is between 0.8 ± 0.1. Figures 5.14–5.18 illustrate classification results from the trained neural networks of various topologies.

In the following figures, a two-layer network topology indicated by [10, 1] implies 10 neurons in the first hidden layer and 1 neuron in the output layer. Similarly, a three-layer neural network topology indicated as [5, 10, 1] implies 5 neurons in the first hidden layer, 10 neurons in the second hidden layer, and 1 neuron in the output layer.

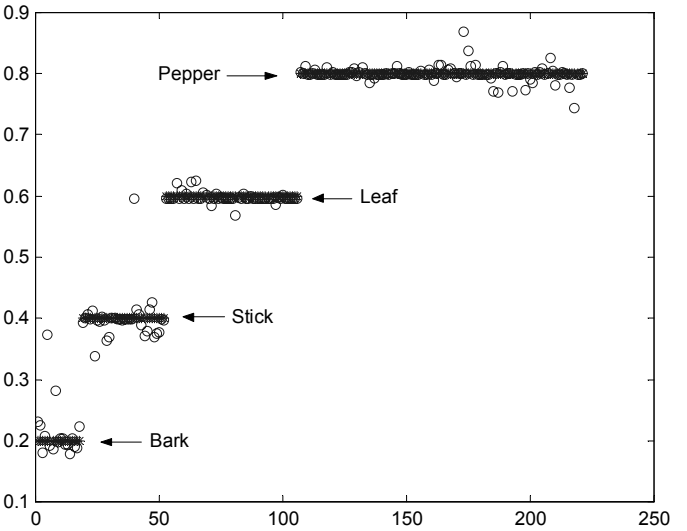


Figure 5.14. Classification performance of [10, 1] topology for Set 1 features

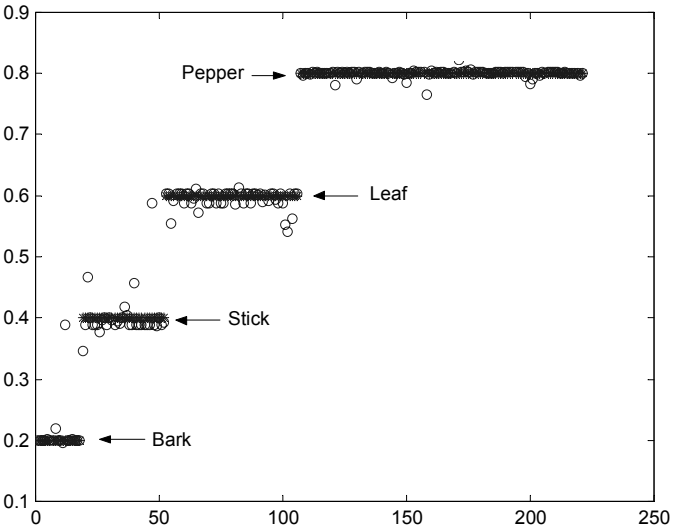


Figure 5.15. Classification performance of [5, 10, 1] topology for Set 1 features

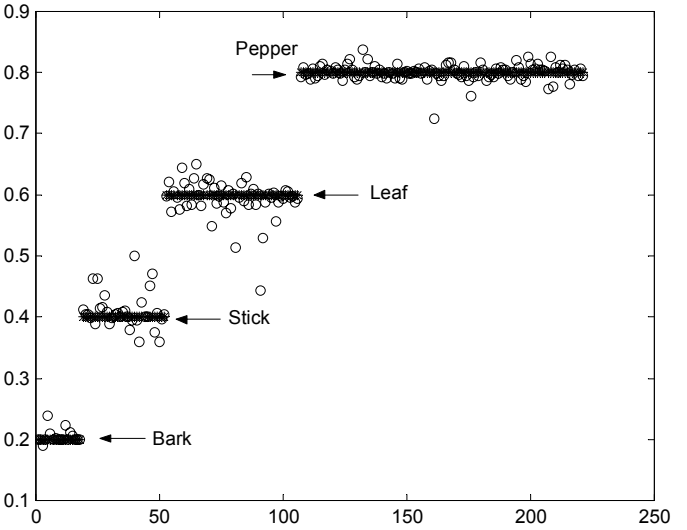


Figure 5.16. Classification performance of [10, 1] topology for Set 2 features

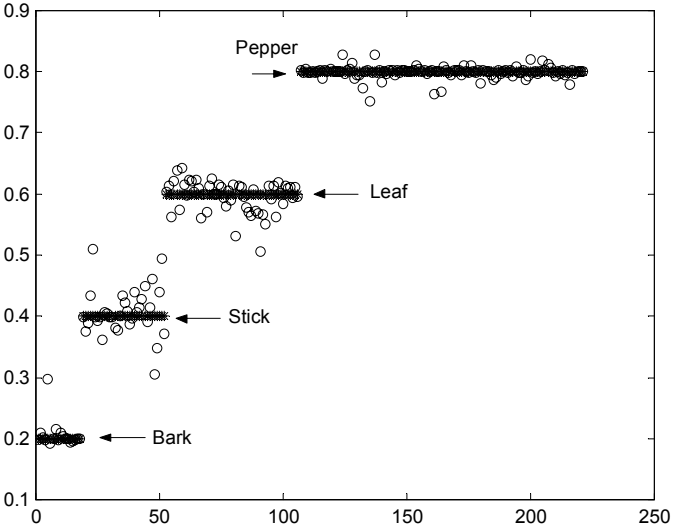


Figure 5.17. Classification performance of [5, 10, 1] topology for Set 2 features

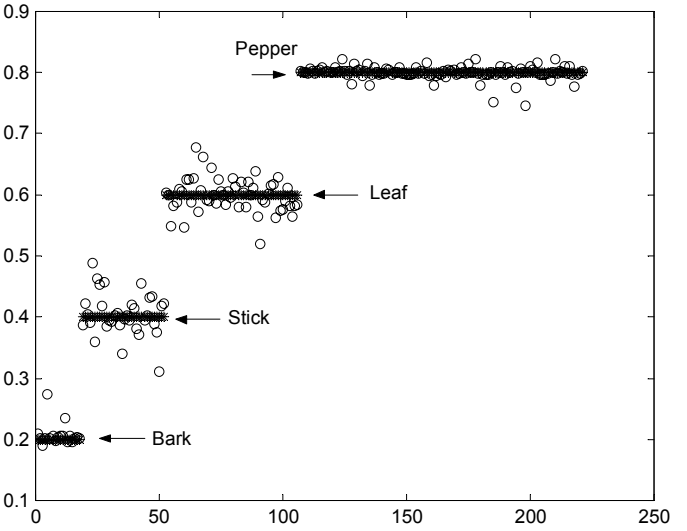


Figure 5.18. Classification performance of [10, 10, 1] topology for Set 2 features

We now refer to Figures 5.14 and 5.15 in which the neural network was trained for Set 1 features listed in Table 5.1. From the results illustrated in Figure 5.14, it is clear that the [10, 1] neural network performance is excellent for the criteria chosen for classification. With the addition of another hidden layer with [5, 10, 1] topology, as illustrated in Figure 5.15, the network performance is even better.

Figures 5.16–5.18 represent the classification results from neural networks trained to Set 2 features. From these results, it is clear that while the performance criteria was indeed met, as in Figure 5.18, the results are not as tightly bound as in Figure 5.15. Results in Figure 5.18 from a [10, 10, 1] topology show improvement over the results from [5, 10, 1] topology illustrated in Figure 5.17. The addition of 5 neurons to the first hidden layer shows marked improvement in performance. From this, we can conclude that either increasing the number of neurons or the addition of another hidden layer may improve the performance. But these additions come at the expense of increased computation and are possibly less attractive to adopt for on-line implementation of trash identification.

It should be clear from this example that classification performance does not necessarily improve with the addition of features. In fact, a simple rule in pattern recognition and classification is to reduce the dimensionality of the input space to a "sufficiently" small number and yet preserve classification performance.

5.8 Practical issues in training

There is no prescribed methodology that predetermines a neural network architecture for a given problem. Some trial and error is required to determine a sufficiently suitable model. The following is a list of several aspects to keep in mind when selecting an appropriate neural network structure.

1. More than one hidden layer may be beneficial for some applications, but in general, one hidden layer is sufficient.

2. The learning rate should be chosen in the open interval $(0, 1)$, since although large η might result in a more rapid convergence, small η avoids overshooting the solution.

3. Training a neural network means creating a general model for an input-output relationship from samples. This model can be applied to new data sets of the problem, that is, can generalize to new data.

4. Overfitting means performing a poor generalization on new data. This happens when the number of parameters (weights) is greater than the number of constraints (training samples).

5. There are no general conclusions about how many neurons should be included in the hidden layer.

6. The choice of initial weights will influence whether the neural network reaches a global or local minimum of the error E, and if so, how quickly it converges (a property of the gradient descent method). The update of the weights w_{ik} depends on both f_k' of the upper layer and the output of the neuron i in the lower layer. For this reason, it is important to avoid choices of the initial weights that would make it likely that either of these quantities is zero.

   Initial weights must not be too large, or the initial input signals will be likely to fall into the region where the derivative of the activation function has a very small value (the saturation region). On the other hand, if the initial weights are too small, the net input to a hidden neuron or output neuron will be close to zero, which also causes extremely slow learning. As a common procedure, the initial weights are chosen at random, either between -1 and 1 or in some other appropriate interval.

7. How long do we need to train a neural network? One could divide a training set into two disjoint subsets: I and II. Use I to train the neural network and use II for testing. During the training, one could compute the errors from II. If these errors decrease, then continue the training. If they increase, then stop the training because the neural network is starting to memorize the set I too specifically and consequently is losing its ability to generalize.

8. What is a good size for the training set? As in statistics, such a question only makes sense when we specify some degree of accuracy of the classification. There is some rule of thumb relating to the number of training samples, the number of weights in a neural network, and the degree of accuracy.

5.9 Exercises and projects

1. Design an artificial neuron to implement the Boolean function AND whose truth table is
$$g(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 = x_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

2. Consider the logical Boolean function "A and not B," symbolically $A \cap B'$, with truth function
$$g(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 = 1 \text{ and } x_2 = 0 \\ 0 & \text{otherwise} \end{cases}$$
Design an artificial neuron to implement g.

3. Using your answer to Exercise 2 and $(w_{AC}, w_{BC}, b_C) = (2, 2, 2)$ from Example 5.1, as values for the two-layer neural network depicted in Example 5.1, show that the network implements the XOR function by verifying that the four possible inputs give the desired outputs.

4. Using your answer to Exercise 2 and $(w_{AC}, w_{BC}, b_C) = (1, -2, -0.5)$, as values for the two-layer neural network depicted in Example 5.1, show that the network also implements the XOR function by verifying that the four possible inputs give the desired outputs.

5. Design an artificial neuron to implement the material implication operator in Boolean logic: A implies $B = A'$ OR B whose truth table is as follows

$x_1 \backslash x_2$	0	1
0	1	1
1	0	1

or in other words,
$$g(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = 1 \text{ and } x_2 = 0 \\ 0 & \text{otherwise} \end{cases}$$

6. Refer to Example 5.3. Verify that the line $x_1 + x_2 = \frac{3}{2}$ in the x_1-x_2 plane partitions the inputs x^q, $q = 1, 2, 3, 4$, into two subsets corresponding to output targets -1, $+1$, respectively.

7. Consider the following training set consisting of bipolar input-output pairs $\{(x^q, y^q),\ q = 1, 2, 3, 4\}$

$x = (x_1, x_2)$	y
$(1, 1)$	1
$(1, -1)$	-1
$(-1, 1)$	-1
$(-1, -1)$	-1

   Find the weight vector that minimizes the error E.

8. Let $f_1(x) = \dfrac{1}{1 + e^{-x}}$ and $f_2(x) = \dfrac{2}{1 + e^{-x}} - 1$

   (a) Compute the derivatives of f_1 and f_2.

   (b) Sketch the graph of f_2.

   (c) Compute the derivative of the hyperbolic tangent function

$$f_3(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

   (d) Verify that

$$f_2(x) = 2f_1(x) - 1 = \frac{1 - e^{-x}}{1 + e^{-x}}$$

   (e) Verify that $f_3(x) = f_2(2x)$.

9. Let $f_1(x) = \dfrac{1}{1 + e^{-x}}$. For $a\, b \in \mathbb{R}$, let $\alpha = b - a$ and $\beta = -a$. Show that the range of the function

$$g(x) = \alpha f_1(x) - \beta$$

   is the open interval (a, b).

10. Given the following function

$$y_1 = 4\sin(\pi x_1) + 2\cos(\pi x_2)$$

   (a) Obtain a set of 20 input-output training data pairs for random variation of (x_1, x_2) in the interval $[-1, 1]$. Train a single-hidden-layered neural network with bipolar sigmoidal functions to the lowest value of tolerance required. You may wish to choose a value of $1.0E - 06$ as a start.

   (b) Obtain a test set of data from the given function with a different seed used in the random number generator. Test the function approximation capability of the neural network. Can the approximation capability be improved by training the neural network with more data? Does over-training cause degradation in performance?

(c) Using the training and test data obtained in (a) and (b), retrain and test a single-hidden-layered neural network with linear functions. Compare the performance of this neural network with that using sigmoidal functions. Can the performance be improved?

(d) For a two-hidden-layered neural network, repeat (a), (b), and (c). Discuss issues pertaining to training time and accuracy in function approximation.

(e) Repeat items (a), (b), (c), and (d) for the function

$$y_2 = \sin(\pi x_1)\cos(0.5\pi x_2)$$

11. A set of six patterns from two classes are specified as follows:

$$\begin{Bmatrix} [3,2]^T, [1,-2]^T, [-2,0]^T \end{Bmatrix} : \text{Class 1}$$
$$\begin{Bmatrix} [2,1]^T, [0,-1]^T, [-1,2]^T \end{Bmatrix} : \text{Class 2}$$

(a) Is this a linearly separable problem?

(b) Design and train a single-hidden-layered neural network with linear activation units.

(c) From the training patterns, generate a set of 20 test patterns that are $\pm 10\%$ of the nominal values. How does the trained neural network perform?

(d) Augment the original set of training patterns with the set of 20 test patterns generated in step (b) and retrain the neural network. For a new set of test patterns, similar to that obtained in step (c), how does the neural network perform?

(e) Show whether the performance can be improved by increasing the number of hidden layers.

(f) Show whether the performance can be improved if the linear activation units were replaced by bipolar or unipolar sigmoidal functions.

(g) What are the consequences of choosing to adopt (e) or (f) in the final design? Which design would you adopt and why?

The student is urged to take on more creative studies on how neural networks can be used in real-world applications. In the following, we provide projects that we consider challenging for the beginner, and hope that such projects can be a motivation to examine more advanced approaches to neural network-based applications. Note that there is quite a bit of diversity and room for exploration.

12. **Project**: The purpose of this project is to develop an aid for the blind in recognizing street-crossing signals. In principle, the system would consist of a miniature camera worn by the individual in a manner that would allow

imaging the street-crossing signal. A real-time neural network processor would then identify the signal/symbol and trigger a voice command that allows the blind to either "stop" or "walk" at the pedestrian crossing. We are required to develop a neural network that can be trained to recognize "WALK" and "STOP" signals commonly found at pedestrian crossings. These signals can be either in the form of words as indicated or symbols such as a "person" or a "hand." You are required to develop the appropriate set of training patterns and train a neural network with four outputs. Each of the outputs should represent the word or symbol used in training. Discuss in detail the technical merits and practical issues in implementing such a system.

13. **Project**: Color is a significant feature in several applications. In the food industry, for example, color is used as an indicator of whether or not the food is of good quality. There is a difference in color between fresh vegetables and not-so-fresh vegetables. The same can be said about meat and other food products. Color is also used in distinguishing between objects such as lemons and lime. As we can see, the possibilities of using color for the purpose of identification of objects are enormous and can be explored in many different applications. We leave the choice of selecting a project to the explorative and inquisitive mind.

    As a motivation to pursue studies in neural networks, in this project however, you are required to develop a neural network that can distinguish between the three colors red, yellow, and green that we commonly see at traffic intersections. Not only is color important, we also recognize the need to identify the direction indicated by the traffic light in terms of the "Left Arrow," the "Right Arrow," the "Up Arrow," and the "Down Arrow." Design a neural network that can identify traffic light conditions. Discuss how such a system may be implemented. Identify and describe specific technical issues that would have to be addressed if such a system were implemented.

14. **Project**: Neural network-based applications are useful wherever there is a need to bridge the gap in communication between humans and machines. Brain-machine interfaces is a new area of scientific exploration where the objective is to make machines respond to human voice, gestures, and thinking. One area that such an application is significant, and can improve the communication ability, is to provide an interface that can translate computer-generated commands to Braille and vice versa. This would be a significant aid to the blind. In this project, we ask that a set of training patterns be selected that adequately represent commonly used words and commands. For example, in using a word processor we choose the "File," "Edit," "Save," and several other menu items to perform file management operations. It would be important to translate these into Braille so that a blind person can use the word processor effectively. Alternatively, if the computer response were "Do you wish to replace an

existing file? Yes or No" such a response would have to be in Braille for the operator to choose the desired option. There is much room in this project to explore the ability of neural networks. The student is urged to be creative.

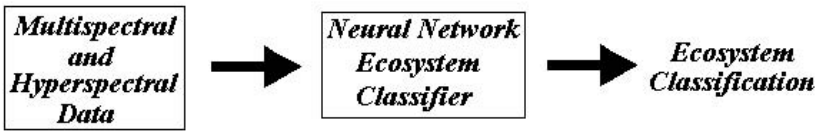
15. **Project**: Several of NASA's planetary missions involve precise and safe landing of the spacecraft on unexplored planetary surface. As part of the Mars Landing Program, NASA's Jet Propulsion Laboratory has a simulated Mars yard where rocks of different sizes are strewn across the yard to emulate the surface of Mars. This simulated environment provides a test bed for examining the capability of a lander to select an appropriate landing site during its descent on to the surface of Mars.

    We can describe briefly the process by which a lander may safely land on the planet Mars. Upon reaching Mars, the Mars orbiter would map the preselected landing site using its diversity of sensors. Due to the distance from which such mapping is done, obviously there is not enough resolution to identify rocks and other debris that may be present in the selected landing site. Once the lander has been deployed, sensors on board the lander would begin to acquire visible and radar images of the landing site and begin to process in real time the "quality" of the landing site. The objective of course is to distinguish "rocky" regions from "smooth" regions and to provide navigational corrections to the lander as it approaches the surface. This requires a real-time adaptive neural network processor that can continuously update itself.

    In this project, we expect the student to be very innovative in how to deal with a simulation-based study of a lander's ability to distinguish "rocky" regions from "smooth" regions. With the availability of digital cameras nowadays, it should be easy to acquire images of different resolution and develop a set of training patterns. We leave the mechanics of data acquisition to the curious and inquisitive mind. The first step however, is to examine the viability of feedforward neural networks to perform adequately. We strongly encourage exploration of other neural network architectures that utilize radial basis functions, principal component analysis, and other techniques that may be more efficient in real-time training and also allow for adaptive capabilities.

16. **Project**: The trend in modern agricultural methods involve the analysis of multispectral and hyperspectral data of agricultural ecosystems obtained from spaceborne and airborne systems. There is quite a lot of research activity in this area at NASA as well as at universities and private institutions. The primary objective in this analysis is to identify the health of the ecosystem by classifying the "state" of the ecosystem. Healthy vegetation, for example, absorbs most of the radiation in the visible spectrum and reflects only a small portion of the visible spectral band. Also, healthy vegetation reflects a large portion of the near-infrared portion of the spectral band. This is indicative of the process of photo-

synthesis and chlorophyll production. Unhealthy plants, however, reflect most of the radiation from the visible spectrum and less from the near-infrared spectral band. The following figure illustrates the overall concept of an ecosystem classifier.



To obtain an estimate of plant health, we compute a value called "normalized difference vegetation index (NDVI)". It is a simple formula using two satellite channel bands. For example, if one band is in the visible region (VIS) say Band 1 (0.58 μm–0.68 μm), and one is in the near-infrared (NIR) say Band 2 (0.725 μm–1.1 μm), then

$$\text{NDVI} = (\text{NIR} - \text{VIS})/(\text{NIR} + \text{VIS})$$

The possible range of values is between -1 and 1, but the typical range is between about -0.1 (NIR $<$ VIS for not very green area) to 0.6 (NIR $>$ VIS for very green area). Healthy vegetation reflects very well in the near-infrared part of the spectrum. Green leaves have a reflectance of 20% or less in the 0.5 to 0.7 micron range (green to red) and about 60% in the 0.7 to 1.3 micron range (near-infrared). Changes in the NDVI values help in monitoring the health of the vegetation over time. NDVI therefore is a very important index to estimate both the short-term as well as the long-term health of agricultural ecosystems.

In this project you are required to train a neural network to classify vegetation. There are several Internet resources that can be useful.

- wwwsgi.ursus.maine.edu/gisweb/spatdb/acsm/ac94014.html
  A technical paper on this website entitled "Classification of Multispectral, Multitemporal, Multisource Spatial Data Using Artificial Neural Networks" provides considerable insight into the possible classifications of vegetation.
- www.geovista.psu.edu/sites/geocomp99/Gc99/072/gc_072.htm
  This website has an excellent technical presentation by David Landgrebe entitled "On information extraction principles for hyperspectral data" and provides a link to download free software.
  A software application program called MultiSpec is available at no cost from dynamo.ecn.purdue.edu/~biehl/MultiSpec/. It contains all the necessary algorithms. Complete documentation of the program is also available. The site also provides sample data sets. The program can be used to extract features required for training a neural network.

Results from a trained neural network can be compared with the results from MultiSpec.

- For the more avid researcher, we provide the following NASA/JPL websites that might be useful in developing a broader perspective of the research required in this field. The reader is urged to follow the links to appropriate websites that discuss NASA's overall mission in Earth and Space Science initiatives.

  earthobservatory.nasa.gov/

  makalu.jpl.nasa.gov/aviris.html

  For graduate student research, the JPL website offers AVIRIS hyperspectral datafree of cost. This website is a great resource for obtaining training and test data.

Chapter 6

NEURAL CONTROL

Neural control refers both to a methodology in which the controller itself is a neural network, and to a methodology in which controllers are designed based on a neural network model of the plant. These two basically different approaches for implementing neural networks in control are referred to as direct and indirect design methods.

6.1 Why neural networks in control

Controlling a dynamical system (a plant) means forcing it to behave in some desired way. Once the desired goal is specified, the design of the controller is dictated by the knowledge about the plant and the data available.

We have seen that fuzzy control is a control method relying on perception-based information expressed in fuzzy logic. This is the case where the available data is in the form of a collection of linguistic "If... then..." rules. In other words, fuzzy control is a mathematical method for implementing control strategies expressed in a natural language. This situation arises mostly in the control of complex systems, a situation that human operators handle well and for which natural language is an appropriate means for describing control strategies.

As its name indicates, neural control refers to another control method when available data are in the form of measurements (observed numerical data) of the plant's behavior. This is the case where information is only in the form of system behavior, either of the real plant or of its simulated model, expressed as input-output measurements. In view of the generality of neural networks as function approximation devices, it is natural to use neural networks in control situations such as this. Specifically, when mathematical models of the plant dynamics are not available, neural networks can provide a useful method for designing controllers, provided we have numerical information about the system behavior in the form of input-output data. In other words, a neural network can be used as a "black box" model for a plant. Also, controllers based on neural networks will benefit from neural networks' learning capability that is suitable

for adaptive control where controllers need to adapt to changing environment, such as for time-variant systems. In practice, neural network controllers have proved to be most useful for time-invariant systems.

Basically, to build a neural network-based controller that can force a plant to behave in some desirable way, we need to adjust its parameters from the observed errors that are the difference between the plant's outputs and the desired outputs. Adjustment of the controller's parameters will be done by propagating back these errors across the neural network structure. This is possible if the mathematical model of the plant is known. When the mathematical model of the plant is not known, we need to know at least an approximate model of the plant in order to do the above. An approximate (known) model of the plant is called an **identified model**. When we use input-output data from the plant to train a neural network to provide an approximate model to the plant, we obtain a **neural network identified model** of the plant. Neural network identified models are used in indirect neural control designs. After a general discussion of inverse dynamics, we will first discuss direct neural control designs and then indirect control.

6.2 Inverse dynamics

An ideal control law describes the **inverse dynamics** of a plant. For a simple example, consider plant dynamics of the form

$$\dot{x}(t) = f(x(t), u(t))$$

A control law for this plant has the form

$$u(t) = g(x(t))$$

Even when a plant possesses inverse dynamics, the solution might not have a closed form, in which case approximations are needed.

In general, it is difficult to check the existence of the inverse dynamics of a plant. However, in the case of linear systems, this existence can be checked easily. In this case, in fact, the existence of inverse dynamics is equivalent to the controllability of the plant, which was discussed in Section 2.3.

Consider, for example, the time-invariant linear system described in discrete time by the equations

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + B\mathbf{u}(k),\ k = 0, 1, 2, ...$$

where A is an $n \times n$ matrix and B is $n \times 1$. Then we have

$$\begin{aligned} \mathbf{x}(k+2) &= A\mathbf{x}(k+1) + B\mathbf{u}(k+1) \\ &= A(A\mathbf{x}(k) + B\mathbf{u}(k)) + B\mathbf{u}(k+1) \\ &= A^2\mathbf{x}(k) + AB\mathbf{u}(k) + B\mathbf{u}(k+1) \end{aligned}$$

$$\begin{aligned}\mathbf{x}(k+3) &= A\mathbf{x}(k+2)+B\mathbf{u}(k+2)\\ &= A\left(A^2\mathbf{x}(k)+AB\mathbf{u}(k)+B\mathbf{u}(k+1)\right)+B\mathbf{u}(k+2)\\ &= A^3\mathbf{x}(k)+A^2B\mathbf{u}(k)+AB\mathbf{u}(k+1)+B\mathbf{u}(k+2)\end{aligned}$$

$$\begin{aligned}\mathbf{x}(k+4) &= A\mathbf{x}(k+3)+B\mathbf{u}(k+3)\\ &= A\left(A^3\mathbf{x}(k)+A^2B\mathbf{u}(k)+AB\mathbf{u}(k+1)+B\mathbf{u}(k+2)\right)+B\mathbf{u}(k+3)\\ &= A^4\mathbf{x}(k)+A^3B\mathbf{u}(k)+A^2B\mathbf{u}(k+1)+AB\mathbf{u}(k+2)+B\mathbf{u}(k+3)\end{aligned}$$

and finally

$$\begin{aligned}\mathbf{x}(k+n) &= A^n\mathbf{x}(k)+A^{n-1}Bu(k)+\cdots+ABu(k+n-2)+Bu(k+n-1)\\ &= A^n\mathbf{x}(k)+WU\end{aligned}$$

where

$$W=\left[\begin{array}{ccccccc} A^{n-1}B & A^{n-2}B & \cdots & A^2B & AB & B\end{array}\right]$$

is the $n\times n$ controllability matrix discussed in Section 2.3, and

$$U=\left[\begin{array}{cccccc} u(k) & u(k+1) & \cdots & u(k+n-2) & u(k+n-1)\end{array}\right]^T$$

If the matrix W is nonsingular, we can compute U directly from W and $\mathbf{x}$:

$$\begin{aligned}U &= W^{-1}\left(\mathbf{x}(k+n)-A^n\mathbf{x}(k)\right)\\ &= \varphi\left(\mathbf{x}(k),\mathbf{x}(k+n)\right)\end{aligned}$$

This is a control law for the system that is derived directly from the plant dynamics by computing the inverse dynamics.

Example 6.1 *Take* $A=\left[\begin{array}{ccc}0&1&0\\0&0&1\\-5&-2&-3\end{array}\right]$ *and* $B=\left[\begin{array}{c}0\\0\\9\end{array}\right]$. *Then*

$$W=\left[\begin{array}{ccc}A^2B & AB & B\end{array}\right]=\left[\begin{array}{ccc}9&0&0\\-27&9&0\\63&-27&9\end{array}\right]$$

and

$$\begin{aligned}U &= \left[\begin{array}{ccc}\frac{1}{9}&0&0\\\frac{1}{3}&\frac{1}{9}&0\\\frac{2}{9}&\frac{1}{3}&\frac{1}{9}\end{array}\right]\left(\mathbf{x}(k+3)-\left[\begin{array}{ccc}-5&-2&-3\\15&1&7\\-35&1&-20\end{array}\right]\mathbf{x}(k)\right)\\ &= \left[\begin{array}{ccc}\frac{1}{9}&0&0\\\frac{1}{3}&\frac{1}{9}&0\\\frac{2}{9}&\frac{1}{3}&\frac{1}{9}\end{array}\right]\mathbf{x}(k+3)-\left[\begin{array}{ccc}-\frac{5}{9}&-\frac{2}{9}&-\frac{1}{3}\\0&-\frac{5}{9}&-\frac{2}{9}\\0&0&-\frac{5}{9}\end{array}\right]\mathbf{x}(k)\\ &= \left[\begin{array}{c}\frac{1}{9}x_1(k+3)+\frac{5}{9}x_1(k)+\frac{2}{9}x_2(k)+\frac{1}{3}x_3(k)\\\frac{1}{3}x_1(k+3)+\frac{1}{9}x_2(k+3)+\frac{5}{9}x_2(k)+\frac{2}{9}x_3(k)\\\frac{2}{9}x_1(k+3)+\frac{1}{3}x_2(k+3n)+\frac{1}{9}x_3(k+3)+\frac{5}{9}x_3(k)\end{array}\right]\\ &= \varphi\left(\mathbf{x}(k),\mathbf{x}(k+n)\right)\end{aligned}$$

When the inverse dynamics of a plant exist, one can try to control the plant by modeling its inverse dynamics.

6.3 Neural networks in direct neural control

As suggested in Section 6.2, suppose we have a plant whose inverse dynamics exists but does not have a closed form. Approximating this inverse dynamics provides a way to control the plant. Of course, approximations of functions (control laws) can be done in many ways. When we use neural networks for modeling inverse dynamics, we are designing direct neural controllers.

Direct design means that a neural network directly implements the controller — that is, the controller is a neural network (see Figure 6.1). The network must be trained as the controller according to some criteria, using either numerical input-output data or a mathematical model of the system.

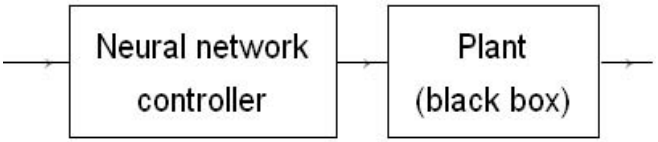


Figure 6.1. Direct design

A natural question that arises in this type of neural control is the selection of the type of neural network needed for the controller. We have seen from the previous chapter on neural networks that there are several types of neural network architectures. Multi-layered perceptron (MLP) neural networks are composed of configurations of simple perceptrons in a hierarchical structure forming a feedforward network. They have one or more hidden layers of perceptrons between the input and output layers. It is permissible to have any prior layer nodes connected to subsequent layer nodes via a corresponding set of weights. Different learning algorithms can be used for MLPs, but the most common ones have been the delta rule and error-backpropagation algorithms discussed previously. These algorithms do work fairly well but they tend to be slow. Faster and more efficient algorithms have been developed [8, 20, 32, 37], and ongoing research is continually discovering further improvements.

6.4 Example: temperature control

In Section 2.7.2, we discussed the development of the classical proportional-integral-derivative (PID) control parameters for a temperature control problem. Here we extend this example to incorporate a neural controller. In a conventional PID control system, the gains are all fixed; while with neural networks, they can change.

6.4.1 A neural network for temperature control

We first demonstrate a very simple neural network control application, where the neural network is intended to function as an ON/OFF controller. The objective here is to train the neural network so that when the measured temperature is higher than desired, the neural controller shuts OFF the furnace, and when the measured temperature is below the desired temperature, the neural controller turns ON the furnace. We must therefore train the neural network to recognize the error between desired and measured temperatures and provide ON/OFF control. Figure 6.2 illustrates the control loop in which a trained neural network acts as the ON/OFF controller.

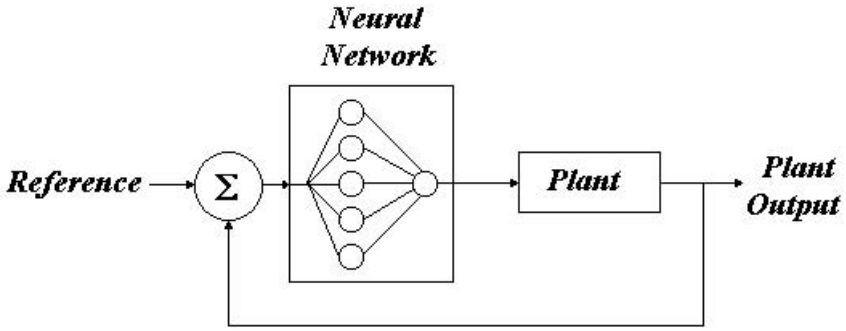


Figure 6.2. Neural network for ON/OFF control

In order to train a neural network, we must first decide what type of patterns we must use so that the trained network will be able to provide the desired control actions for a wide range of variability in the error signal. Typically, we wish to select a pattern that is oscillatory in nature and has some damping associated with it. We therefore select a damped sinusoid and expect a trained neural network to provide generalization for many other variations in the error signal. For this example, we generate a training pattern vector $p(i)$, for $i = 1, 50$. We also generate a target vector $t(i)$, for $i = 1, 50$ so that each value in $p(i)$ has a corresponding target value $t(i)$. Since this is an ON/OFF controller design, we set all positive values of $p(i)$ to have a target $t(i) = +1$, and for all negative values of $p(i)$ we set $t(i) = -1$. This is similar to a relay switch. Note that since the pattern vector is assumed to represent the error signal that is the difference "reference minus measured," training the neural network to a relay characteristic would provide the necessary ON/OFF controller design.

First we generate the training pattern, in this case a damped sinusoid, from which we develop the appropriate target vectors. We set the target as $+1$ for all positive values of the damped sinusoid and to -1 for all negative values of the damped sinusoid. This is illustrated in the following MATLAB code.

```
w=2*pi;
time=0;
for i=1:50,
```

```
    p(i)=(exp(-time))*sin(w*time);
        time=time+0.1;
    if (p(i)>0.0)
        t(i)=1;
    elseif (p(i)<0.0)
        t(i)=-1;
end
end
```

Figure 6.3 illustrates the damped sinusoid generated from the preceding MATLAB code.

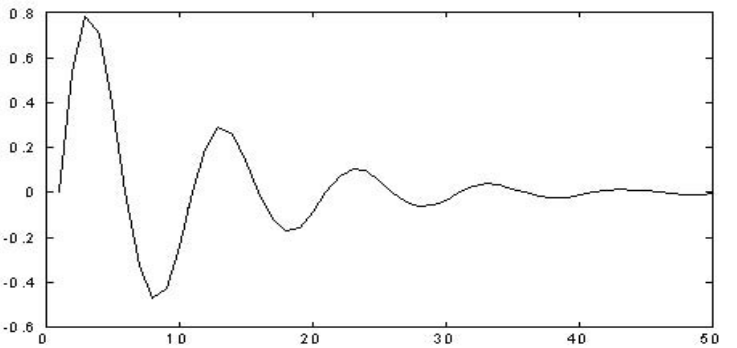


Figure 6.3. Damped sinusoidal pattern for neural network training

Figure 6.4 illustrates the relay characteristic obtained by plotting the pattern vector $p(i)$ versus the target $t(i)$. The neural network will be trained to reproduce this relay characteristic.

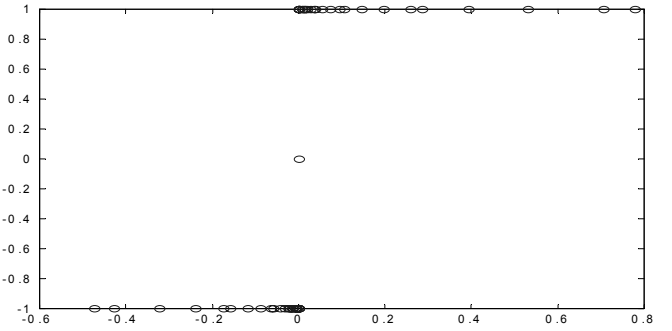


Figure 6.4. Plot of pattern vector versus target that represents a relay characteristic

In our example, a two-layer feedforward network is created. The topology of the network will comprise a hidden layer with some specified number of neurons, and an output layer with as many neurons as we desire to represent

the number of outputs we wish to generate. For example, in a system with one binary output, we will need one output neuron. Selecting the number of neurons in the hidden layers is by trial and error. To start with, we choose the first layer (the hidden layer) to have 10 neurons and the second layer (the output layer) to have 1 neuron. All 10 neurons in the hidden layer are assigned to have tangent-sigmoid nonlinear (squashing) functions. In MATLAB this is the function "`tansig`". The output neuron is assigned a linear output function "`purelin`." Note that "`purelin`" provides both positive and negative outputs as opposed to "`poslin`" which provides only positive output values. We will use "`trainlm`," which is the **Levenberg-Marquardt optimization algorithm**, as the default neural network training algorithm. The default backpropagation learning function and mean-squared error performance function are "`learngdm`" and "`mse`," respectively.

The MATLAB setup that provides the desired neural network topology is as follows:

```
net = newff([min(p) max(p)],[10 1],{'tansig','purelin'},
        'trainlm','learngdm','mse');
```

We also need to specify the number of epochs over which training is performed and the mean-square error. In this example we set `mse` $= 0.179$ and the training period for 10,000 epochs. Note that the number of epochs is the maximum number of training cycles that the neural network is allowed to train with the goal to meet the mean-squared error criterion. This section of the MATLAB code is as follows:

```
net.trainParam.epochs = 10000;
net.trainParam.goal = 0.179;
```

Before proceeding to the training phase, it would be prudent to initialize all the weights and biases of the neural network. For this we use MATLAB's initialization code as follows:

```
net=init(net);
net.iw{1,1}
net.b{1}
```

Now we are ready to train the neural network. The following MATLAB code performs training.

```
net = train(net,p,t);
```

It is always good to check how well the neural network has been trained by resimulating the training function, in this case the relay characteristic. The following simulation and plot commands allow visual confirmation of the trained neural network response. Figure 6.5 plots the original function with "`o`" and the output "`y1`" of the trained function with "$*$."

```
y1=sim(net, p);
plot(p,t,'o',p,y1,'*');
```

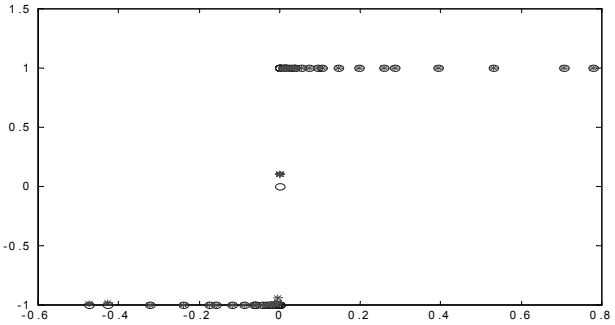


Figure 6.5. Original function 'o' versus trained function '$*$'

Referring to Figure 6.5, it is clear that the neural network has learned the relay characteristic very well and that no further retraining is necessary. If, on the other hand, the trained response were unsatisfactory, we may have had to increase the number of neurons in the hidden layer or choose another topology, perhaps one with an additional hidden layer, as a possible choice for a neural network that would provide better learning capabilities. We must note from this discussion that there is no unique design for a neural network.

MATLAB allows us to view the error surface created by plotting the sum-square error as a function of the weights and biases of a single neuron. This surface gives some indication as to how well the neural network has been trained. The segment of the MATLAB code used to view the error surface is shown below, and the error surface is illustrated in Figure 6.6.

```
wv=-1:0.1:1;
bv=-2.5:.25:2.5;
es=errsurf(p,t,wv,bv,'tansig');
plotes(wv,bv,es,[60, 30]);
```

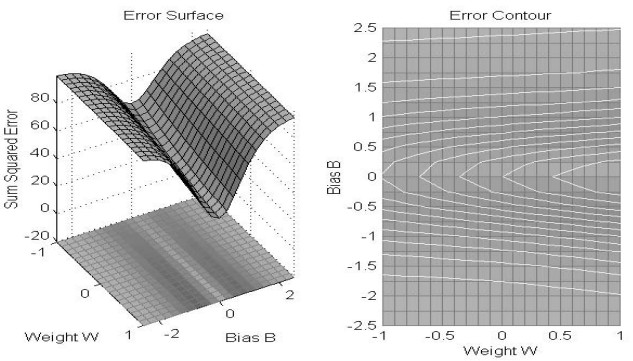


Figure 6.6. Error surface

We are now in a position to test the performance of the neural network to provide ON/OFF control in a closed-loop configuration. To test this performance, we generate an arbitrary testing function as shown below.

```
w=16*pi;
time=0;
for i=1:100,
p2(i)=16*(exp(-time))*sin(w*time);
time=time+0.1;
end
```

We can then simulate the network response using the following MATLAB command and obtain a plot of the testing function in relation to the trained function. Figure 6.7 illustrates the trained and test results.

```
y2 = sim(net,p2)
plot(p,t,'go',p2,y2,'b*');
```

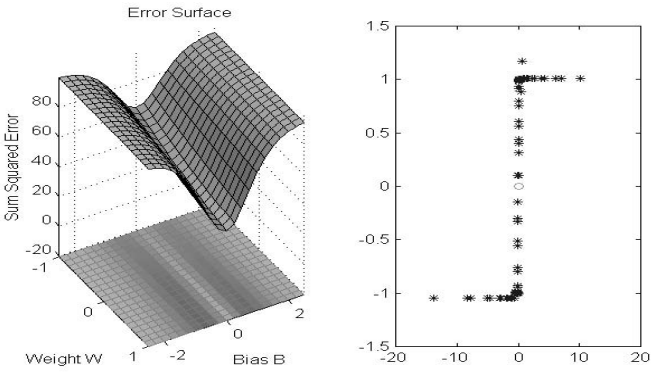


Figure 6.7. Simulated ON/OFF for a test function

6.4.2 Simulating PI control with a neural network

We now extend the previous example to include proportional and integral control functions. Once again, we generate an exponentially damped sinusoid pattern vector $p(i)$, $i = 1, 50$. Note that this pattern vector is assumed to represent the error between a reference and the actual output of the plant. A damped sinusoidal response is typical of plant behavior for which a controller needs to be designed. As before, we use the following MATLAB code to generate the pattern vector $p(i)$.

```
w=2*pi;
time=0;
for i=1:50,
    p(i)=(exp(-time))*sin(w*time);
```

```
    time=time+0.1;
end
```

We obtain a plot of the generated training pattern vector as shown in Figure 6.8.

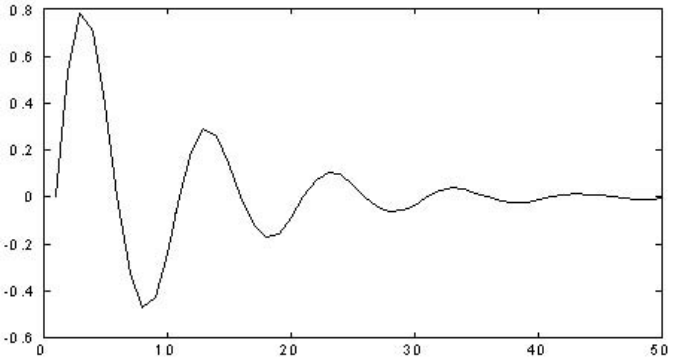


Figure 6.8. Training pattern vector

We then compute and plot the gradient of the training pattern vector, shown in Figure 6.9.

```
dp = gradient(p);
plot(dp,'*');
```

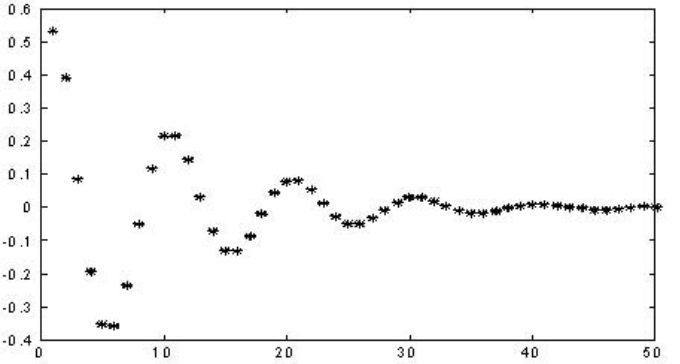


Figure 6.9. Gradient of training pattern vector

The gradient of the pattern vector allows us to develop a prototype proportional gain function and the pattern vector itself can be used to develop a prototype integral gain function. These are obtained in MATLAB as follows:

```
for i=1:50
tprop(i)=10.0*abs(dp(i));
tint(i)=exp(-abs(p(i)));
end
```

Plots of the proportional gain versus the gradient and integral gain versus the pattern vector are then obtained, as illustrated in Figures 6.10 and 6.11.

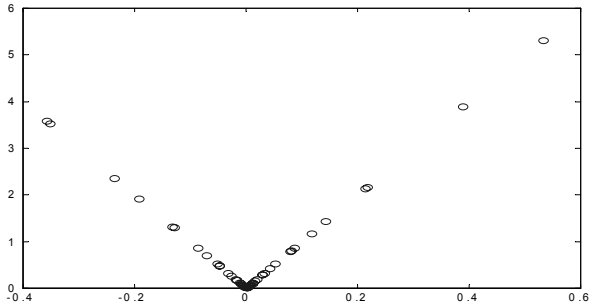


Figure 6.10. Proportional gain versus gradient of the pattern vector

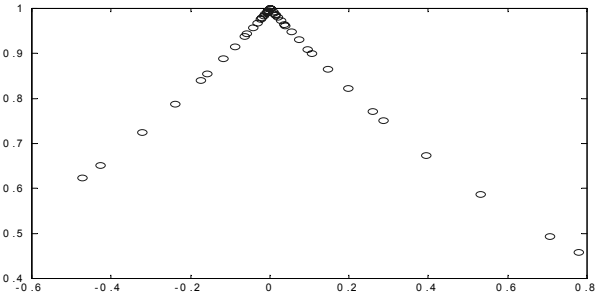


Figure 6.11. Integral gain versus pattern vector

We then set up two feedforward neural networks according to the MATLAB neural network function "`newff`" syntax:

net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF).

In our example, a two-layer feedforward network is created. The first hidden layer has 10 "`tansig`" neurons, and the output layer has one "`poslin`" neuron. We once again use "`trainlm`" as the default neural network training function. The default backpropagation learning function "`learngdm`" and mean-squared error performance function "`mse`" are used as before. The first neural network "`net1`" is used to control the proportional gain and the second neural network "`net2`" is used to control the integral gain.

```
net1 = newff([min(dp) max(dp)],[10 1],{'tansig','poslin'},
             'trainlm','learngdm','mse');
net2 = newff([min(p) max(p)],[10 1],{'tansig','poslin'},
             'trainlm','learngdm','mse');
```

As noted in the previous example, it is best to initialize all neural network parameters prior to training. We initialize both the neural networks as follows:

```
net1 = init(net1);
net1.iw{1,1}
net1.b{1}
net2 = init(net2);
net2.iw{1,1}
net2.b{1}
```

The training period is set up for 10,000 epochs along with a mean-squared error goal of 0.0001.

```
net1.trainParam.epochs = 10000;
net1.trainParam.goal = 0.0001;
net2.trainParam.epochs = 10000;
net2.trainParam.goal = 0.00001;
```

The networks "`net1`" and "`net2`" can now be trained to the proportional gain and integral gain functions developed previously.

```
net1 = train(net1,dp,tprop);
net2 = train(net2,p,tint);
```

The two trained neural networks "`net1`" and "`net2`" are resimulated to check how well the networks have learned the control functions, and the plots shown in Figures 6.12–6.15 are obtained.

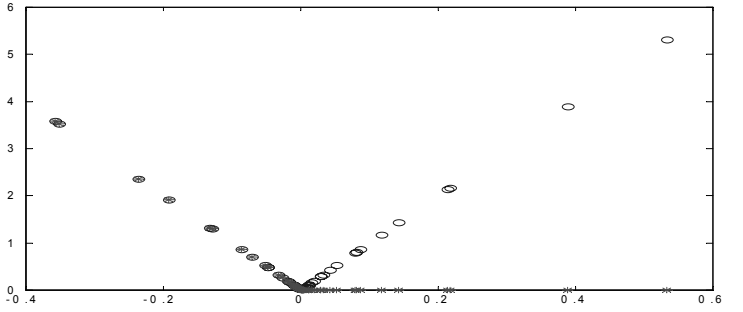


Figure 6.12. Proportional gain versus gradient for "`net1`"

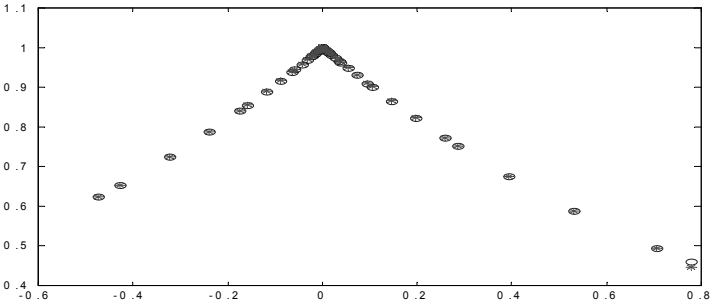


Figure 6.13. Integral gain versus pattern vector for "net2"

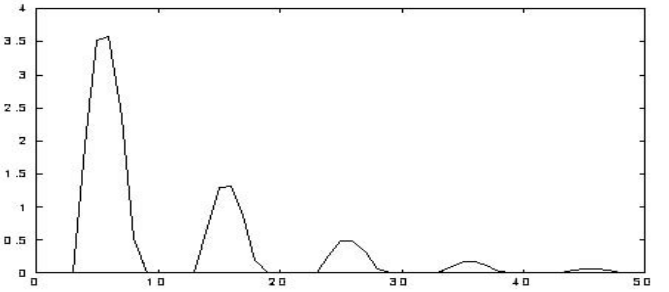


Figure 6.14. Variation in proportional gain to the training pattern

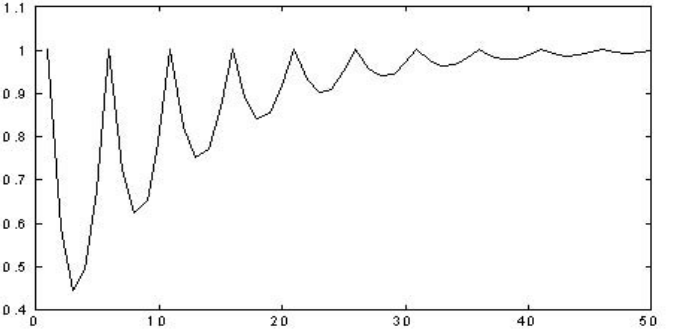


Figure 6.15. Variation in integral gain to the training pattern

We now plot the resulting error surfaces shown in Figures 6.16 and 6.17.

```
wv=-1:0.1:1; bv=-2.5:.25:2.5;
es11=errsurf(dp,tprop,wv,bv,'tansig');
plotes(wv,bv,es11,[75, 25]);
es12=errsurf(p,tint,wv,bv,'tansig');
plotes(wv,bv,es12,[75, 25]);
```

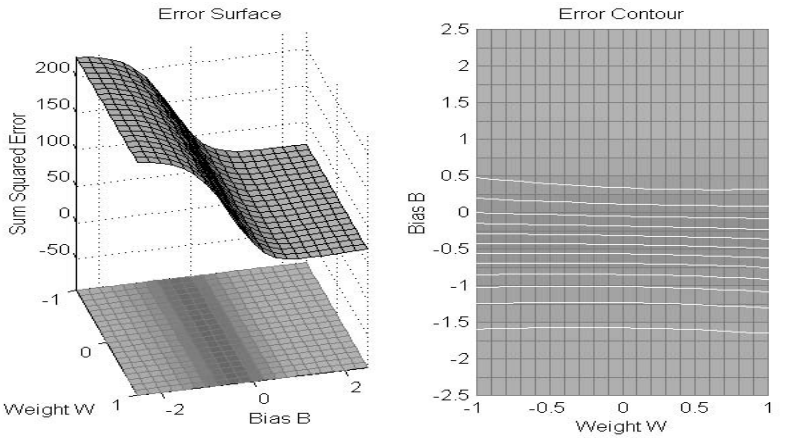


Figure 6.16. Proportional control network error surface

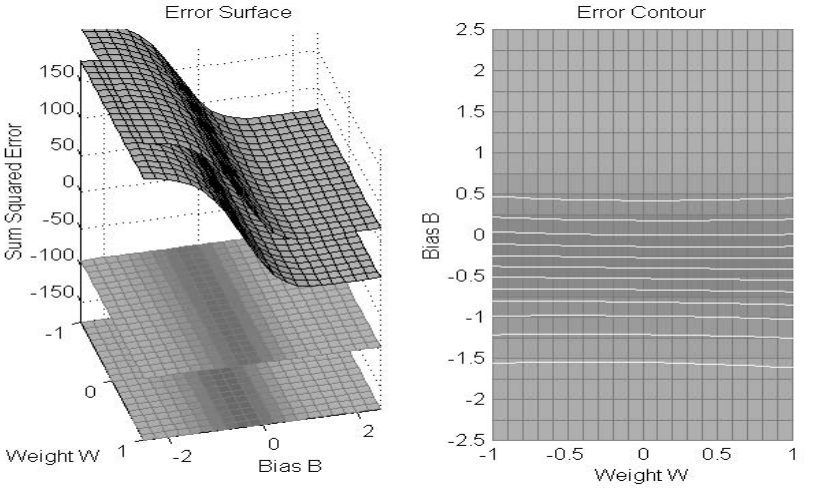


Figure 6.17. Integral control network error surface

From the results of training the two neural networks, we conclude that both neural networks have learned the control functions well. We do however need to test the neural network performance and see whether the networks can perform well as controllers. For this, an arbitrary testing function is now simulated to see how well the neural networks are able to provide proportional and integral control.

```
w=1*pi;
time=0;
for i=1:100,
```

```
    p2(i)=5*(exp(-time))*sin(w*time);
    time=time+0.1;
end;
dp2=gradient(p2)
```

The plots in Figures 6.18 and 6.19 illustrate the generated test pattern vector and its gradient.

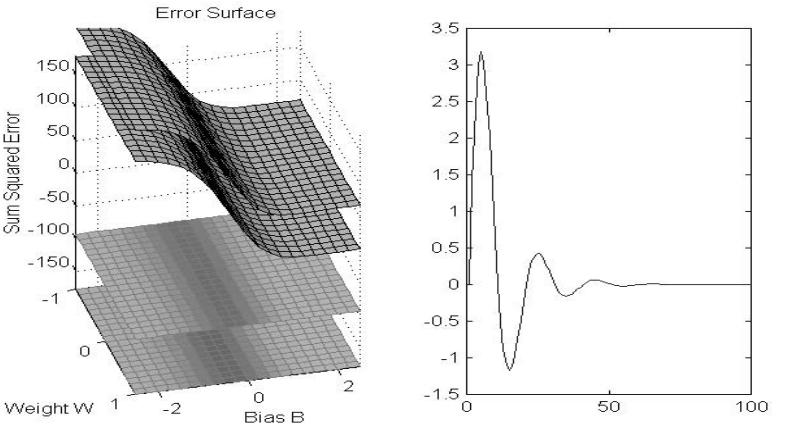


Figure 6.18. Test pattern vector

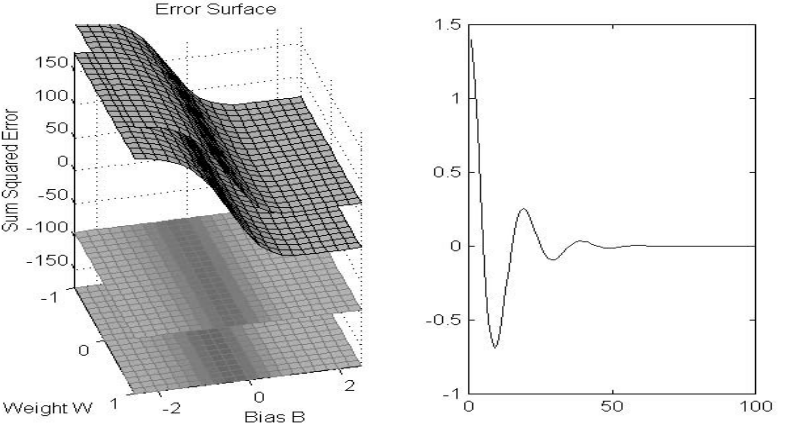


Figure 6.19. Gradient of test pattern vector

We now simulate the trained networks to an arbitrary input and plot the resulting variations in proportional and integral gain. These are shown in Figures 6.20 and 6.21.

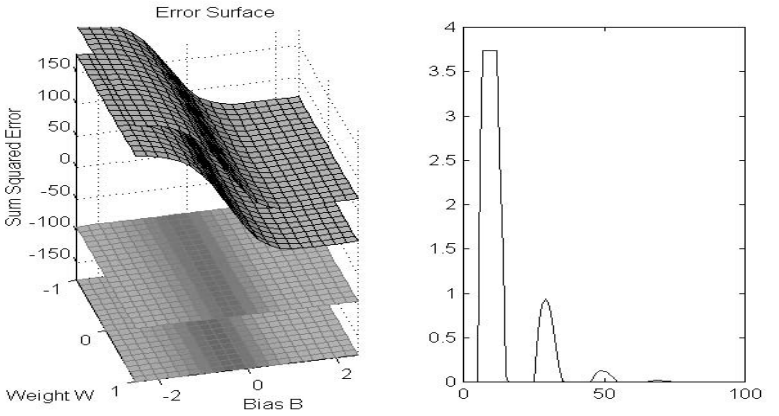


Figure 6.20. Variation in proportional gain to test pattern

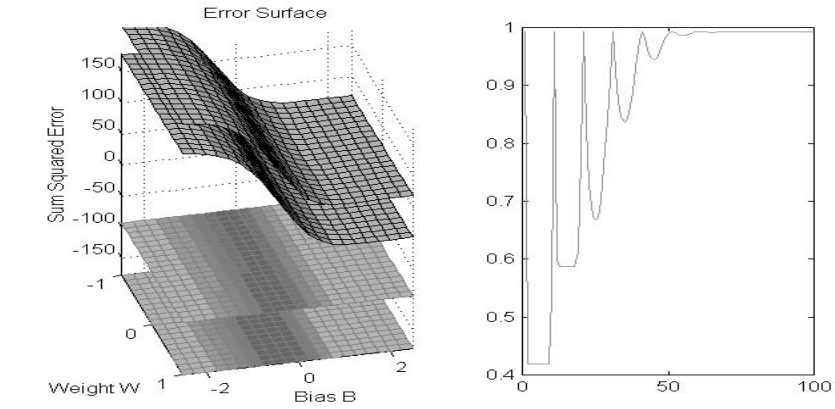


Figure 6.21. Variation in integral gain to test function

From this example, we can see the benefits of using neural control in comparison with conventional PID control. Both proportional and integral gains are variable and can be scaled externally to provide the desired response. We leave as an exercise the development of a single neural network that can provide both proportional and integral control. Choose an appropriate feedforward neural network topology in which two output neurons can each provide proportional and integral gain control.

6.5 Neural networks in indirect neural control

Indirect neural control design is based on a neural network model of the system to be controlled. In this case, the controller itself may not be a neural network, but it is derived from a plant that is modeled by a neural network.

This is similar to standard control in that a mathematical model is needed, but here the mathematical model is a neural network.

Indirect neural control designs involve two phases. The first phase consists of identifying the plant dynamics by a neural network from training data — that is, system identification. In the second phase, the control design can be rather conventional even though the controller is derived, not from a standard mathematical model of a plant, but from its neural network identified model. Since the identified neural network model of the plant is nonlinear, one way to design the controller is to linearize its identified neural network model and apply standard linear controller designs. Another way is through "instantaneous linearization," as described in Section 6.5.3.

6.5.1 System identification

The system identification phase can be carried out by using neural networks. Recall that a controller and the plant that it controls bear an inverse relationship, and that a primary goal in the development of an autonomous system is to build a controller whose behavior bears an inverse relationship to the plant. In particular, system identification is necessary for establishing a model on which the controller design can be based. The general problem of system identification of nonlinear systems is daunting and, lacking sufficient theoretical results, much needs to be done on a case-by-case basis. It is one of the areas where the application of neural networks is promising.

The following figure depicts the general problem of system identification of nonlinear systems. The parameters of the identification model are estimated as the model changes over time, so that the difference between the plant output and the model output is minimized.

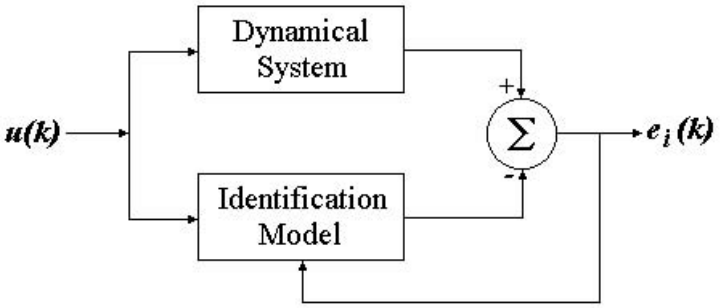


Figure 6.22. System identification model

The idea is for the identification process to produce a model of the dynamical system with no prior knowledge of the dynamics of the system. This is referred to as **black box modeling**. The learning algorithm that is used to train the network is commonly a version of backpropagation, as discussed in Section 5.5. Of course, the identification model may be reliably trained only for the data it experiences and may not produce the desired output for new data. However,

in practice, system identification by neural networks works remarkably well for reasonably well-behaved dynamical systems.

A neural network that is trained to classify plant behavior by observing the output of the plant must be trained based on the errors that might occur in the plant response compared to a desired response. The error function that results from training the neural network must very closely approximate the inverse behavior of the plant. A neural network can therefore be used to approximate control laws that govern the controlling inputs to the plant. Neural networks are particularly appropriate for system identification when the mathematical model is unknown and the behavior of the system is known only in the form of sample data. Neural networks can be used for nonlinear systems, and they can also be used for optimal control.

The first step in system identification is experimentation with the plant to develop physical insight. A good experiment will result in a set of data that includes responses to input over the entire range of operation of the plant. This data set will be used to train the model.

The next step is to select a neural network model. This choice includes both selecting the inputs to the network and selecting an internal network topology. The two most widely used families of models for this purpose are multi-layer perceptrons and the so-called radial basis function neural networks, a family of neural network models that we do not treat in this book.

The model can be represented as

$$y(t) = g(\mathbf{x}(t,\theta),\theta) + e(t)$$

where $\mathbf{x}(t,\theta)$ is a regression vector and θ is a vector containing the weights. One form the regression vector can take is

$$\mathbf{x}(t,\theta) = [y(t),...,y(t-n),u(t-d),...,u(t-d-m,),e(t,\theta),...,e(t-k,\theta)]^T$$

The choice of regressor will depend in part on knowledge of the plant behavior.

The model is then trained with the data that was obtained from the experiment. After the training, the model should be validated to check whether it meets the necessary criteria in terms of the intended use of the model. If the results are not completely satisfactory, different neural network models can be tried. If the data collected is not sufficient, it may be impossible to develop a satisfactory model and one must start over, carrying out a new experiment to obtain additional training data. It could be, for example, that not all actual inputs that affect behavior of the system were recognized, and these must be added to the experiment and to the model before achieving the desired result.

When system identification is implemented as part of the controller, the controller is known as an **adaptive controller**. Such controllers are designed to control systems whose dynamical characteristics vary with time. A more detailed exposition of system identification can be found in advanced texts on neural network control, such as [30] and [54].

6.5.2 Example: system identification

As indicated previously, the goal in system identification is to obtain a satisfactory model of the nonlinear system by observing the input-output behavior of the system. Given a nonlinear process, we need to determine the functional relationship between output and the input. The system itself is a black box whose internal structure is not available. All we can observe is what goes into the black box and what comes out of the black box. For such identification, neural networks have been found to yield excellent results.

In this section, we discuss a step-by-step procedure for developing a neural network-based identification of a nonlinear system. This example, originally appearing on pages 731–733 in [40], has been adapted here with permission. Let us consider a nonlinear system of the form:

$$y_p(k+1) = f\left(y_p(k), y_p(k-1), \cdots\right) + u(k)$$

where $f\left(y_p(k), y_p(k-1), \cdots\right)$ is the nonlinear system to be identified, and $u(k)$ is the input applied to the system. The function $f\left(y_p(k), y_p(k-1), \cdots\right)$ represents the internal structure of the nonlinear system that is used in obtaining the next state $y_p(k+1)$. Clearly, therefore, if the response $y_p(k)$ of the nonlinear system can be sampled over a range of possible system operating conditions — that is, for a range of inputs $u(k) \in [\min, \max]$ over which stable plant operation can be guaranteed, then mapping the input-output behavior onto a neural network becomes a viable choice for identifying the nonlinear system. In this example, a nonlinear system is given by

$$y_p(k+1) = \frac{y_p(k)\left(y_p(k-1)+2\right)\left(y_p(k)+2.5\right)}{8.5 + \left(y_p(k)\right)^2 + \left(y_p(k-1)\right)^2} + u(k)$$

which is of the form

$$y_p(k+1) = f\left(y_p(k), y_p(k-1)\right) + u(k)$$

For this system, stable plant operation is guaranteed for $u(k) \in [-2, 2]$. The objective in this example is to train a feedforward neural network with error backpropagation such that the outputs of the trained neural network and the actual plant are the same for the specified inputs. We can now proceed to demonstrate the identification process using MATLAB.

Step 1. We first need to obtain the input-output pairs of data that can be used to train a neural network. In this example, we simulate the behavior of the nonlinear system and develop the appropriate training data. For a real-world process however, this amounts to obtaining measured data of the input and output of the plant. Since we are computing the next state, $y_p(k+1)$, we need to initialize the parameters for $y_p(k)$ and $y_p(k-1)$. The following MATLAB code provides this required first step.

```
yp(1)=0; yp(2)=0; %yp(1) = yp(k-1); yp(2) = yp(k)
```

Step 2. Generate an input vector "u" in the range $[-2, 2]$. Our objective in this step is to develop a vector of random inputs for which we can generate a

corresponding vector of outputs. Select a suitable number of vector elements. In this example we have chosen to develop a vector of 301 random input elements.

```
u=rands(1, 301)*2; % u(k) ∈ [−2, 2]
```

Step 3. Simulate the response of the nonlinear system using the random input vector generated in Step 2 and create the input-output pairs of training data.

```
for k=2:301
yp(k+1)=yp(k)*(yp(k-1)+2)*(yp(k)+2.5)/(8.5+yp(k)^2
        +yp(k-1)^2)+u(k);
out(k-1)=(yp(k+1)-u(k))/20; %Output training data
in(k-1)=yp(k)/20; %Input training data
end;
```

Notice that the output and input training data have been scaled, in this case, by a factor of 20. Scaling is needed when using sigmoidal functions so that we create an adequate spread in the "squashed" data. The user must experiment with this and select a suitable scaling factor that will help in the neural network convergence. Over-scaling or under-scaling can significantly affect the network convergence properties.

Step 4. Set up the input-output data vectors for neural network training. Here the input data "`plantin`" is set up as pairs of the form

$$\left[\left\{\begin{array}{l} y_p(k) \\ y_p(k+1) \end{array}\right\}\right]$$

namely,

$$\left[\left\{\begin{array}{l} in(1) \\ in(2) \end{array}\right\}, \left\{\begin{array}{l} in(2) \\ in(3) \end{array}\right\}, \left\{\begin{array}{l} in(3) \\ in(4) \end{array}\right\}, \cdots, \left\{\begin{array}{l} in(298) \\ in(299) \end{array}\right\}, \left\{\begin{array}{l} in(299) \\ in(300) \end{array}\right\}\right]$$

and the corresponding output "`plantout`" for each input pair is set up as

$$[out(1), out(2), out(3), \cdots, out(299)]$$

These are the pattern and target vectors that the neural network uses for training. The following MATLAB code performs this setup.

```
plantin=[in(1:299); in(2:300)]; plantout=out(1:299);
```

Step 5. As a first attempt to obtain a suitable trained neural network, here we need to choose the network topology. The number of layers and the number of neurons in each layer are at issue. Choosing a large number of neurons obviously increases the computations and hence affects the time to converge. For a trial, we choose 1 neuron in the first hidden layer, 10 neurons in the second hidden layer, and 1 output neuron. The selection of the number of output neurons is based upon the number of outputs in the system. Select the appropriate activation function that characterizes the input-output behavior. For this example, a bipolar activation function is necessary and hence the MATLAB "`tansig`" function is chosen in all three layers. The "`trainlm`" algorithm is a network training

function that updates weight and bias values according to Levenberg-Marquardt optimization. Other training algorithms must be explored to determine their effectiveness in meeting desired convergence criteria.

```
net = newff(minmax(nni),[1 10 1],{'tansig' 'tansig' 'tansig'},
        'trainlm','learngdm','mse')
```

Step 6. Initialize the chosen neural network structure.

```
net = init(net);
net.iw{1,1}
net.b{1}
```

Step 7. Set the number of training epochs and the desired error tolerance. You may have to increase the number of epochs if the convergence tolerance is not met within the number specified. On the other hand, you may have to increase the tolerance to obtain convergence. Nonconvergence generally is due to ill-conditioned data. As such, some experimentation is needed in scaling the training data to obtain a "suitable" set.

```
net.trainParam.epochs = 500;
net.trainParam.goal = 0.0005;
```

Step 8. Train the neural network for the input-output data.

```
net=train(net,plantin,plantout);
```

Step 9. Obtain the neural network response to the random input used for training.

```
trainedout=sim(net,plantin);
```

Step 10. Compare the results of the actual plant response with that produced by the neural network for the same input.

```
plot(plantout, 'b'); %Actual plant output
hold on;
plot(trainedout, ':k'); %Trained neural network output
grid;
axis([0, 300, -0.1, 0.3]);
xlabel('Time Step'); ylabel('Plant (solid) NN Output (dotted)');
```

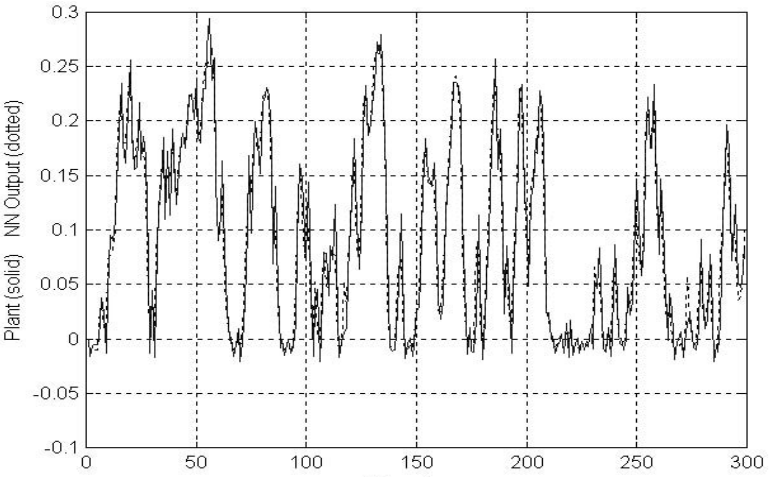


Figure 6.23. Plant and trained neural network response to the same random input

At this stage, as shown in Figure 6.23, we have a trained neural network that we hope will perform well when normally occurring input data is applied to it. If the same input applied to the plant is applied to the neural network, and both produce the same output, then we have identified the plant behavior and obtained a model of the plant. The next step therefore involves setting up the input that would be normally applied to the plant and to simulate the behavior of both the plant and the corresponding trained neural network.

Step 11. Generate a vector of input data and simulate both the original plant response and the trained neural network. The following MATLAB code generates a specified input $u(k)$ and simulates the behavior of the nonlinear system and of the trained neural network.

```
yp(1)=0.0; yp(2)=0.0;out1(1)=0; out1(2)=0;
for k=2:500
   if (k<=200)u(k)=2.0*cos(2*pi*k*0.01);
     else
   u(k)=1.2*sin(2*pi*k*0.05);
   end;
yp(k+1)=yp(k)*(yp(k-1)+2)*(yp(k)+2.5)/(8.5+yp(k)^2
       +yp(k-1)^2)+u(k);
out1(k)=yp(k)/20;
out1(k-1)=yp(k-1)/20;
nnout(k+1)=20*sim(net,[out1(k);out1(k-1)])+u(k);
end;
 plot(yp, 'b');
 hold on;
 plot(nnout, ':k');
 grid;
 axis([0, 500, -4.0, 10.0]);
```

```
xlabel('Time Step'); ylabel('Plant (solid) NN Output (dotted)');
```

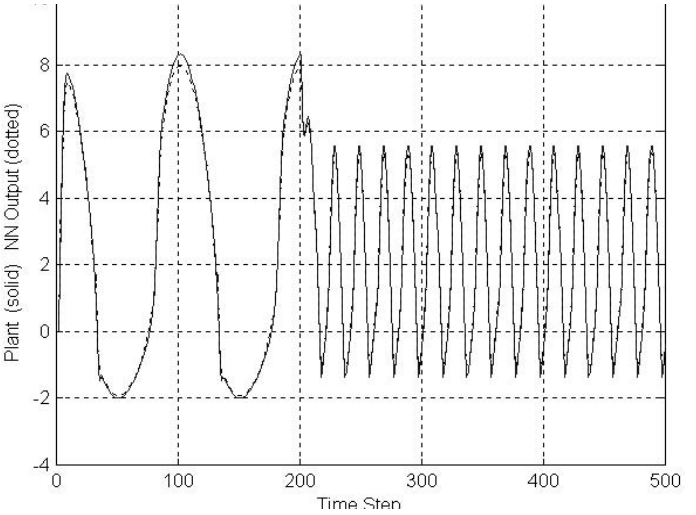


Figure 6.24. Plant and trained neural network response to specified input

Step 12. If the results are satisfactory, as illustrated in the simulation results of Figure 6.24, then we have identified the nonlinear system and we can use the trained neural network for control purposes. If however the results are not satisfactory, some amount of experimentation would be required in selecting the appropriate number of neurons in each of the hidden layers and/or increasing the number of hidden layers, choosing a proper scaling factor, and by choosing a different training algorithm. The reader is urged to conduct such experimentation in order to obtain a better feel for how such parameters affect neural network performance and training characteristics.

In the example discussed, it is clear that the neural network performs very well in identifying the "unknown" plant. Several models for identification are discussed in the reference cited on page 219, at the beginning of this example. We strongly recommend that the reader obtain further information regarding system identification issues from this and other sources.

6.5.3 Instantaneous linearization

The most common approach to the control of nonlinear systems is to approximate the system by a linear system, in the region of concern, and then to design a controller for this linear system. Neural network model structures produce discrete nonlinear models, and as mentioned earlier, one way to design the controller is to linearize its identified neural network model and apply standard linear controller designs. However, linearization of these discrete nonlinear models can also be carried out at each sampling time by a process called **instantaneous linearization**. This process extracts a linear model from the

nonlinear neural network model at each sampling time.

Assume that a neural network input-output model of the system to be controlled is described as a function of past outputs $y(t-i)$ and past inputs $u(t-d-i)$ in the form

$$y(t) = g(\mathbf{x}(t))$$

where the vector

$$\mathbf{x}(t) = [y(t-1), ..., y(t-n), u(t-d), ..., u(t-d-m)]^T$$

defines the state of the system. At time $t = \tau$, linearize the function

$$g = g(x_1, ..., x_{n+m+1})$$

around the current state $\mathbf{x}(\tau)$ to obtain the approximate model

$$\tilde{y}(t) = -a_1\tilde{y}(t-1) - \ \ldots\ - a_n\tilde{y}(t-n) + b_0\tilde{u}(t-d) + \ \ldots\ + \tilde{u}(t-d-m)$$

where

$$\begin{aligned}
\tilde{y}(t-i) &= y(t-i) - y(\tau-i) \\
\tilde{u}(t-i) &= u(t-i) - u(\tau-i) \\
a_i &= -\left.\frac{\partial g(\mathbf{x}(t))}{\partial x_i}\right|_{t=\tau} \quad \text{for } 1 \le i \le n \\
b_i &= -\left.\frac{\partial g(\mathbf{x}(t))}{\partial x_{n+i+1}}\right|_{t=\tau} \quad \text{for } 0 \le i \le m
\end{aligned}$$

For a multi-layer perceptron (MLP) network with n_x inputs, one hidden layer of n_h tanh units, and a linear output

$$y(t) = \sum_{j=1}^{n_h} W_j \tanh\left(\sum_{k=1}^{n_x} w_{kj}x_k(t) + w_{0j}\right) + W_0$$

the derivative of the output with respect to input $x_i(t)$ is calculated in accordance with

$$\frac{\partial g(\mathbf{x}(t))}{\partial x_i(t)} = \sum_{j=1}^{n_h} W_j w_{ji}\left(1 - \tanh^2\left(\sum_{k=1}^{n_x} w_{jk}x_k(t) + w_{j0}\right)\right)$$

where

$$(x_1, ..., x_{n_x}) = (y_1, ..., y_n, u_0, u_1, ..., u_m)$$

The approximate model can also be expressed as

$$y(t) = -\sum_{i=1}^{n} a_i y(t-i) + \sum_{i=0}^{m} b_i u(t-i) + \left(y(\tau) + \sum_{i=1}^{n} a_i y(\tau-i) - \sum_{i=0}^{m} b_i u(\tau-i)\right)$$

where

$$\zeta(\tau) = y(\tau) + \sum_{i=1}^{n} a_i y(\tau - i) - \sum_{i=0}^{m} b_i u(\tau - i)$$

is the bias term. The approximate model may thus be interpreted as a linear model affected by a constant disturbance $\zeta(\tau)$, depending on the current operating point.

To apply the instantaneous linearization technique to the design of controllers, a linear model is extracted from a neural network model of the system at each sampling time and a linear controller is designed. The control design is based on the certainty equivalence principle — that is, the design block assumes the extracted linear model is a perfect description of the system. One can regard this as a gain-scheduling controller with an infinite schedule.

One appealing feature of the linearization technique is that essentially any linear control design can be incorporated in the process. Linearization results in the bias term have to be compensated for, of course. This can be achieved by introducing integral action into the controller, in which case other constant disturbances will be compensated for as well.

Depending on the character of the nonlinearities of the system, and the selected reference trajectory, the linear model can be considered valid only within a relatively short time period. One must be careful to choose controller designs not violating the limits of the approximation. Instantaneous linearization can be applied to deterministic or stochastic models, but we have discussed only the deterministic case. We refer you to [54] for more details on instantaneous linearization.

6.6 Exercises and projects

1. A nonlinear system is defined by $y(k) = f(y(k-1)) + [u(k-1)]^3$, where

$$f(y(k-1)) = \frac{y(k-1)}{1 + y(k-1)^2}$$

is the system to be identified. Develop a backpropagation neural network that can identify the nonlinear system. Examine the performance to the following input

$$u(k) = \begin{cases} 2e^{-0.02\pi k} & \text{if } 0 \le k \le 50 \\ 10e^{-0.01\pi k} \sin(0.2\pi k) & \text{if } 50 < k \le 150 \end{cases}$$

Compare the performance of the network using two of MATLAB's training algorithms, namely, `trainlm`, the Levenberg-Marquardt optimization, and `traingd`, the gradient descent optimization techniques.

2. A nonlinear chemical process is specified as $y(t) = f(y(t-1), y(t-2)) + 0.1\,(u(t-1))^2$, where $f(y(t-1), y(t-2)) = 0.5y(t-1) - 0.1y(t-1)y(t-2)$ is the system to be identified. Develop a backpropagation neural network that can identify the nonlinear system.

(a) Test the neural network performance to an input specified by $u(t) = \begin{cases} 0.2e^{-0.02\pi k}\sin(0.2\pi k) & \text{if } 0 \le k \le 100 \\ 0.2e^{-0.02\pi k}\sin(0.2\pi k) + 0.8e^{(-0.02\pi k)}\cos(0.2\pi k) & \text{if } 100 < k \le 300 \end{cases}$

(b) Simulate the performance of the trained neural network with $u(t)$ as specified above, and with additive Gaussian noise $\eta(t) = N(0, 0.04)$. Discuss the performance.

3. A nonlinear process is characterized by a single input and two outputs as

$$\begin{aligned} y_1(t) &= f_1(y_1(t-1), y_1(t-2), y_2(t-1), y_2(t-2)) + u(t) \\ y_2(t) &= f_2(y_1(t-1), y_1(t-2), y_2(t-1), y_2(t-2)) + u(t) \end{aligned}$$

where

$$f_1(y_1(t-1), y_1(t-2), y_2(t-1), y_2(t-2)) = \left(1.85 + 0.24e^{-y_1(t-1)^2}\right) y_2(t-1) - \left(0.35 + 0.34e^{-y_1(t-2)^2}\right) y_2(t-2)$$

$$f_2(y_1(t-1), y_1(t-2), y_2(t-1), y_2(t-2)) = \left(1.35 + 0.28e^{-y_2(t-1)^2}\right) y_1(t-1) - \left(0.65 + 0.25e^{-y_2(t-2)^2}\right) y_1(t-2)$$

(a) Determine through simulations the range of $u(t) \in [\min, \max]$ for which the nonlinear system is stable.

(b) Develop a backpropagation neural network that identifies f_1 and f_2.

(c) Choosing a variable frequency sinusoidal input within the range determined in part (a), discuss the performance of the trained neural network.

4. For the magnetic levitation exercise in Chapter 4, we are required to replace the fuzzy controller with a suitable neural network controller. The equation of motion for the steel ball and the equation governing variation in the current in the electrical circuit are reproduced here for convenience. For the steel ball,

$$M\frac{d^2z}{dt^2} = Mg - F$$

and for the electrical circuit,

$$L\frac{di}{dt} + Ri = V$$

where L is the coil inductance and R is the coil resistance. The coupling equation where the force F is related to the current i is given by

$$F = k_m\frac{i^2}{z^2}$$

Assume that the ball position is measured using an optical sensor. Use the following set of parameters in your model setup.

M	Mass of steel ball	20 milligrams (mg)
k_m	Magnetic constant	2.058×10^{-4} N (m/ A)^2
R	Coil resistance	0.92 Ohms (Ω)
L	Coil inductance	0.01 millihenry (mH)
i	Coil current	$[0, 3]$ Amps (A)
V	Coil voltage	$[0, 5]$ Volts (V)
g	Gravitational constant	9.80665 m/ s^2
z	Ball position [min, max]	$[3, 7]$ cm

(a) Through simulation, obtain a suitable set of training parameters to train a backpropagation neural network. Use the specified range of parameters for the current i, the voltage V, and the distance z over which the ball is allowed to move in the vertical direction.

(b) Test the neural network performance for various disturbances that affect the ball position.

(c) Compare your results with the fuzzy controller developed in Chapter 4. Does neural control offer any advantages over fuzzy control? Explain why, or why not.

5. For the liquid level control problem in Chapter 4, we wish to replace the fuzzy controller with a neural controller.

(a) Develop a suitable set of neural network training parameters using the same set of system parameters used for the fuzzy controller design.

(b) Compare the neural network performance with the fuzzy control performance.

6. A neural controller is desired to control the ball position in the ball and beam problem of Chapter 4. For convenience, the nonlinear equations are specified here again along with the system parameters:

$$\left[\frac{J}{r^2} + M\right] \ddot{R} + Mg \sin\alpha - mR\left(\dot{\alpha}\right)^2 = 0$$

The beam angle α may be approximated by a linear relationship $\alpha = \frac{D}{L}\theta$. These equations form the set of coupled equations for the system. Using the following system parameters

M mass of the ball 0.2 kg
R radius of the ball 0.02 m
D lever arm offset 0.03 m
g gravitational acceleration 9.8 m/ s^2
L length of the beam 1.5 m
J the moment of inertia of the ball $2.0e^{-6}$ kg m^2

(a) Simulate the nonlinear equations for small perturbations around the equilibrium to obtain a suitable set of neural network training parameters.

(b) Test the performance of the controller for disturbances acting on the ball.

(c) Comment on the neural controller performance in relation to the fuzzy controller.

Chapter 7

FUZZY-NEURAL AND NEURAL-FUZZY CONTROL

So far we have discussed two distinct methods for building controllers: fuzzy and neural. Often the choice of method is dictated by the data available on the plant involved. If the data are pairs of numbers, we may turn to a neural method, and if the data are rules, fuzzy methods may be appropriate. Neural methods provide learning capability, whereas fuzzy methods provide flexible knowledge-representational capability. Integrating these two methodologies, in control in particular and in intelligent technologies in general, can lead to better technologies that take advantage of the strengths of each methodology and at the same time overcome some of the limitations of the individual techniques.

In this chapter, we discuss methods for combining neural and fuzzy methods to build controllers. There are many ways in which these methods can be combined. Complicated controllers can have different component problems, each of which may require different types of processing, but such complex situations are beyond the scope of this book. Within a single component, there are still basically two ways that fuzzy and neural technologies can be combined. In one direction, fuzzy logic can be introduced into neural networks to enhance knowledge representation capability of conventional neural networks. This can be done by introducing fuzzy concepts within neural networks — that is, at the levels of inputs, weights, aggregation operations, activation functions, and outputs. Standard mathematical models for neurons can, for example, be changed to "fuzzy neurons" with t-norms and t-conorms used to build aggregation operations. This leads to a **fuzzy-neural system** with which one can present fuzzy inputs and develop an analog of the conventional backpropagation algorithm for training.

In the other direction, neural networks can be used in fuzzy modeling and control to provide fuzzy systems with learning capabilities. These methods lead

to a **neural-fuzzy system** — a fuzzy system represented as a modified neural network, resulting in a fuzzy inference system that is enhanced by neural network capabilities. Fuzzy systems are generally more "user friendly" than neural systems because their behavior can be explained based on fuzzy rules fashioned after human reasoning. Although fuzzy logic can encode expert knowledge directly using rules with linguistic labels, it usually takes a lot of time to design and tune the membership functions that quantitatively represent these linguistic labels, and applications of pure fuzzy control systems are restricted mainly to those fields where expert knowledge is available and the number of input variables is small. Neural network learning techniques can automate this process and substantially reduce development time and cost while improving performance. Neural networks are also used to preprocess data and to extract fuzzy control rules from numerical data automatically, as well as to tune membership functions of fuzzy systems. In this chapter, we first address some issues of fuzzy-neural systems for control problems, and then look at neural-fuzzy systems. Our primary focus is on adaptive neuro-fuzzy systems for control.

7.1 Fuzzy concepts in neural networks

Fuzzy logic concepts can be incorporated into neural network structure at any level. Recall that a Mamdani fuzzy rule is of the form

$$\text{"If } x \text{ is } A \text{ then } y \text{ is } B\text{"}$$

and a Sugeno fuzzy rule is of the form

$$\text{"If } x \text{ is } A \text{ then } y \text{ is } f(x)\text{"}$$

where A and B are fuzzy sets or products of fuzzy sets, and f is a real-valued function. If A is the product of fuzzy subsets $A_1, ..., A_n$ of U_i, $i = 1, 2, ..., n$, then

$$A : \prod_{i=1}^{n} U_i \to [0,1] : (u_1, ..., u_n) \mapsto \min\{A_1(u_1), ..., A_n(u_n)\}$$

and for $x = (x_1, ..., x_n)$, "x is A" stands for "x_1 is A_1 and x_2 is A_2, ..., x_n is A_n," and f is a real-valued function on $\mathbb{R}^n$. The fuzzy inference THEN is implemented most commonly by minimum, the "Mamdani implication," or by the product. Rules are combined by a fuzzy OR, namely by some t-conorm — usually maximum.

One modification of a neural network structure is to replace some or all components of a neuron by fuzzy logic operations. Such a neuron is called a **fuzzy neuron**. For example, if addition (as an aggregation operation) in a neuron is replaced by minimum, we have a **min-fuzzy neuron**. A neuron that uses maximum as an aggregation operation is a **max-fuzzy neuron**.

A neural network with fuzzy neurons becomes a **multi-layer fuzzy-neural network** (Figure 7.1). Note that conventional neural networks are used to

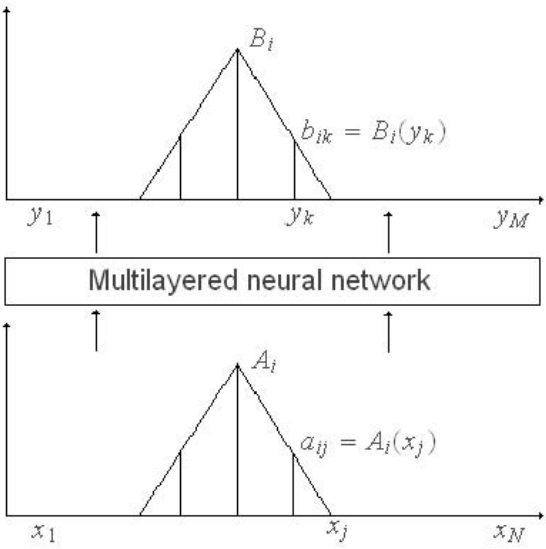


Figure 7.1. Neural network with fuzzy neurons

approximate functions from numerical input-output data (x_i, y_i), $i = 1, 2, ..., n$. Fuzzy-neural networks are a more general computational structure with which function approximation can be extended to linguistic data.

Consider a set of fuzzy rules of the form

$$R_i : \text{If } \mathbf{x} \text{ is } A^i \text{ then } \mathbf{y} \text{ is } B^i \tag{7.1}$$

$i = 1, ..., n$, where A^i and B^i are products of fuzzy sets

$$\begin{aligned} A^i(\mathbf{x}) &= \left(\prod_{j=1}^{N} A_j^i\right)(\mathbf{x}) = \bigwedge_{j=1}^{N} A_j^i(x_j) \\ B^i(\mathbf{y}) &= \left(\prod_{k=1}^{M} B_k^i\right)(\mathbf{y}) = \bigwedge_{k=1}^{M} B_k^i(y_k) \end{aligned}$$

Each rule in (7.1) can be interpreted as a training pattern for a multi-layer neural network, where the antecedent part of the rule is the input to the neural network, and the consequent part of the rule is the desired output of the neural net. Equation (7.1) therefore would constitute a

- single-input, single-output (SISO) system if $N = M = 1$.
- multi-input, single-output (MISO) system if $N > 1$, $M = 1$.
- multi-input, multi-output (MIMO) system if $N > 1$, $M > 1$.

The derived training set consists of input-output pairs (A^i, B^i). For a MIMO fuzzy system, the derived training set can be written in the form

$$(\{A_j^i\}_{j=1}^N, \{B_k^i\}_{k=1}^M),\ i = 1, ..., n$$

For a MISO system, this simplifies to

$$(\{A_j^i\}_{j=1}^N, B_i),\ i = 1, ..., n$$

and for a SISO system, the derived training set can be written in the form

$$(A_1, B_1), ..., (A_n, B_n)$$

We also need to incorporate defuzzification operations into neural networks. These operations, when necessary, take place at the final level of the neural network and are the same operations as those discussed in Chapter 3.

7.2 Basic principles of fuzzy-neural systems

Rules express the behavior of a system. They allow us to specify the output, given the input. This property is particularly useful in that we can define the desired output to obtain the "best" performance of the system being controlled. For a PID controller, this gives the flexibility to specify the various gains, namely, the proportional gain, the derivative gain and the integral gains to achieve the desired performance.

In this section we look at how fuzzy "If... then..." rules express the input-output behavior of fuzzy-neural systems. Knowledge of rules allows us to examine appropriate trainable neural network architectures. We show how the structure of fuzzy rules can be transformed into neural networks, giving rise to fuzzy-neural systems.

There are two main approaches to implement fuzzy "If... then..." rules by a standard-error backpropagation network — a method first proposed by Umano and Ezawa [75] in 1991, and a modification by Uehara and Fujise [74] in 1992 that uses a finite number of α-level sets to represent fuzzy numbers. In 1993 Jang [35] extended these methods toward the development of an adaptive network fuzzy inference system (ANFIS) that has spawned numerous applications. We will discuss ANFIS later in this chapter.

In the method proposed by Umano and Ezawa, a fuzzy set is represented by a finite number of its membership values. Suppose we have a set of fuzzy rules of the form

$$R_i : \text{If } x \text{ is } A_i \text{ then } y \text{ is } B_i$$

$i = 1, ..., n$, where A_i and B_i are fuzzy sets. Let the interval $[a_1, a_2]$ contain the support of all the A_i, plus the support of any other functions we might have as input to the system. Also, let $[b_1, b_2]$ contain the support of all the B_i, plus the support of any other functions we can obtain as outputs from the system, $i = 1, ..., n$.

We can work with a finite number of membership values by taking positive integers $M \geq 2$ and $N \geq 2$, and setting

$$\begin{aligned} x_j &= a_1 + \frac{j-1}{N-1}(a_2 - a_1) \\ y_k &= b_1 + \frac{k-1}{M-1}(b_2 - b_1) \end{aligned}$$

for $1 \leq j \leq N$ and $1 \leq k \leq M$. This gives us a discrete version of the continuous training set consisting of the input-output pairs

$$(A_i(\mathbf{x}), B_i(\mathbf{y})) = ((A_i(x_1), ..., A_i(x_N)), (B_i(y_1), ..., B_i(y_M)))$$

for $i = 1, ..., n$, and the fuzzy-neural network turns into an N-input and M-output standard network that can be trained by the generalized delta rule (see page 180).

Example 7.1 Assume our fuzzy rule base consists of three rules

R_1 : If x is small then y is negative

R_2 : If x is medium then y is about zero

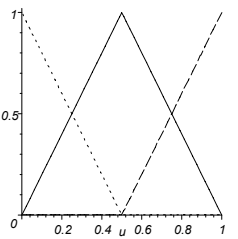
R_3 : If x is big then y is positive

where the membership functions of fuzzy terms are defined by

$$\mu_{small}(u) = A_1(u) = \begin{cases} 1 - 2u & \text{if} \quad 0 \leq u \leq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

$$\mu_{medium}(u) = A_2(u) = \begin{cases} 1 - 2|u - 1/2| & \text{if} \quad 0 \leq u \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\mu_{big}(u) = A_3(u) = \begin{cases} 2u - 1 & \text{if} \quad 1/2 \leq u \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

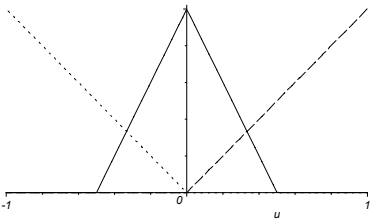


Fuzzy sets *small (dots), medium (solid)*, and *big* (dashed)

$$\mu_{negative}(u) = B_1(u) = \begin{cases} -u & \text{if} \quad -1 \leq u \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mu_{about\ zero}(u) = B_2(u) = \begin{cases} 1 - 2|u| & \text{if} \quad -1/2 \leq u \leq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

$$\mu_{positive}(u) = B_3(u) = \begin{cases} u & \text{if} \quad 0 \leq u \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



Fuzzy sets negative (dots), *about zero (solid), and positive (dashed)*

The fuzzy training set derived from this rule base can be written in the form

{(small, negative), (medium, about zero), (big, positive)}

Let $[0, 1]$ contain the support of all the fuzzy sets we might have as input to the system. Also, let $[-1, 1]$ contain the support of all the fuzzy sets we can obtain as outputs from the system. Let $M = N = 5$ and

$$\begin{aligned} x_j &= (j-1)/4 \\ y_k &= -1 + (k-1)2/4 = -1 + (k-1)/2 = -3/2 + k/2 \end{aligned}$$

for $1 \leq j \leq 5$ and $1 \leq k \leq 5$. Substituting for i and j, we get

$$\begin{aligned} \mathbf{x} &= (x_1, x_2, x_3, x_4, x_5) = (0, 0.25, 0.5, 0.75, 1) \\ \mathbf{y} &= (y_1, y_2, y_3, y_4, y_5) = (-1, -0.5, 0, 0.5, 1) \end{aligned}$$

A discrete version of the continuous training set consists of three input-output pairs for use in a standard backpropagation network

$$\begin{aligned} (A_1(\mathbf{x}), B_1(\mathbf{y})) &= \left(\begin{pmatrix} 1 & 0.5 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0.5 & 0 & 0 & 0 \end{pmatrix}\right) \\ (A_2(\mathbf{x}), B_2(\mathbf{y})) &= \left(\begin{pmatrix} 0 & 0.5 & 1 & 0.5 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \end{pmatrix}\right) \\ (A_3(\mathbf{x}), B_3(\mathbf{y})) &= \left(\begin{pmatrix} 0 & 0 & 0 & 0.5 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 0.5 & 1 \end{pmatrix}\right) \end{aligned}$$

To demonstrate the distinction between a regular neural network and a fuzzy-neural network, we briefly consider the crisp case of a neural network that was discussed in detail in Chapter 5. Consider a simple neural net as in Figure 7.2. All signals and weights are real numbers. The two input neurons do not change the input signal, so their output is the same as their input. The signal x_i interacts with the weight w_i to produce the product

$$p_i = w_i x_i, \quad i = 1, 2$$

The input information p_i is aggregated by addition to produce the input

$$net = p_1 + p_2 = w_1 x_1 + w_2 x_2$$

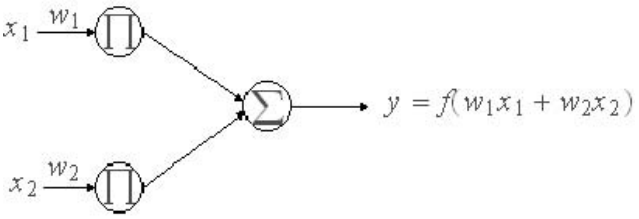


Figure 7.2. Standard neuron

to the neuron. The neuron uses its transfer function f, often a sigmoidal function $f(x) = \frac{1}{1+e^{-x}}$, to compute the output

$$y = f(net) = f(w_1x_1 + w_2x_2)$$

In the crisp case, we use addition and multiplication operators to perform aggregation of the input signals (Figure 7.2). However, this is not possible in the case of fuzzy numbers. If we employ operators like a t-norm or a t-conorm to combine the incoming data to a neuron, we obtain what we call a *hybrid neural net*. These modifications lead to a fuzzy-neural architecture based on fuzzy arithmetic operations. A hybrid neural net may not use multiplication, addition, or a sigmoidal function because the results of these operations are not necessarily in the unit interval.

Definition 7.1 *A **hybrid neural net** is a neural net with crisp signals and weights and a crisp transfer function for which*

- *The signals and weights x_i and w_i, both of which lie in the interval $[0,1]$, can be combined using a t-norm, t-conorm, or some other continuous operation.*
- *The results p_1 and p_2 can be aggregated with a t-norm, t-conorm, or any other continuous function from $[0,1]$ to $[0,1]$.*
- *The transfer function f can be any continuous function from $[0,1]$ to $[0,1]$.*

*A processing element of a hybrid neural net is called a **fuzzy neuron**.*

We emphasize that all inputs, outputs, and weights of a hybrid neural net are real numbers taken from the unit interval $[0,1]$.

Example 7.2 (AND) Figure 7.3 illustrates an AND fuzzy neuron. The signals x_i and weights w_i are combined by a triangular conorm S to produce the output

$$p_i = S(w_i, x_i), \quad i = 1, 2$$

The input information p_i is aggregated by a triangular norm T to produce the output

$$y = AND(p_1, p_2) = T(p_1, p_2) = T(S(w_1, x_1), S(w_2, x_2))$$

of the neuron. If $T = \min$ and $S = \max$, then the AND neuron realizes the max-min composition $y = \min\{w_1 \vee x_1, w_2 \vee x_2\}$.

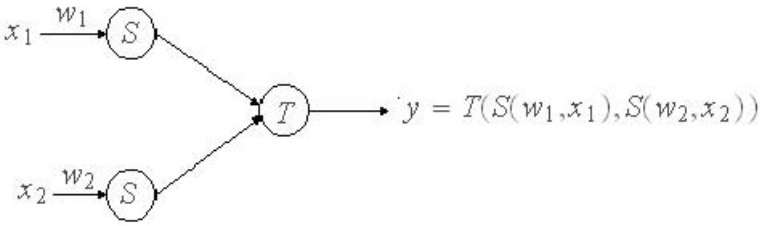


Figure 7.3. AND fuzzy neuron

Example 7.3 (OR) Figure 7.4 illustrates an OR fuzzy neuron. The signal x_i and weight w_i are combined by a triangular norm T to produce

$$p_i = T(w_i, x_i), \quad i = 1, 2$$

The input information p_i is aggregated by a triangular conorm S to produce the output

$$y = OR(p_1, p_2) = S(p_1, p_2) = S(T(w_1, x_1), T(w_2, x_2))$$

of the neuron. If $T = \min$ and $S = \max$, then the OR neuron realizes the

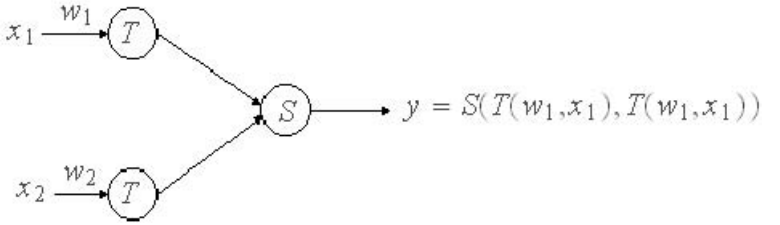


Figure 7.4. OR fuzzy neuron

max-min composition $y = \max\{w_1 \wedge x_1,\ w_2 \wedge x_2\}$.

7.3 Basic principles of neural-fuzzy systems

In order to process fuzzy rules by neural networks, it is necessary to modify the standard neural network structure appropriately. Since fuzzy systems are universal approximators, it is expected that their equivalent neural network representations will possess the same property. As stated earlier, the reason to

represent a fuzzy system in terms of a neural network is to utilize the learning capability of neural networks to improve performance, such as adaptation of fuzzy systems. Thus, the training algorithm in the modified neural networks should be examined.

7.3.1 Adaptive network fuzzy inference systems

To illustrate the use of neural networks for fuzzy inference, we present some successful adaptive neural network fuzzy inference systems, along with training algorithms known as ANFIS. These structures, also known as adaptive neuro-fuzzy inference systems or adaptive network fuzzy inference systems, were proposed by Jang [35]. It should be noted that similar structures were also proposed independently by Lin and Lee [40] and Wang and Mendel [77]. These structures are useful for control and for many other applications.

To fix the ideas, consider the problem of graphically representing the way fuzzy control is achieved in the Sugeno-Takagi model. For a simple example, consider a fuzzy rule base consisting of only two rules:

$$\begin{array}{l} \mathrm{R}_1\text{: If } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } B_1 \text{ then } y = f_1(\mathbf{x}) \\ \mathrm{R}_2\text{: If } x_1 \text{ is } A_2 \text{ and } x_2 \text{ is } B_2 \text{ then } y = f_2(\mathbf{x}) \end{array}$$

where A_i and B_i are fuzzy sets and

$$\begin{aligned} f_1(\mathbf{x}) &= z_{11}x_1 + z_{12}x_2 + z_{13} \\ f_2(\mathbf{x}) &= z_{21}x_1 + z_{22}x_2 + z_{23} \end{aligned}$$

Recall that when numerical input $\mathbf{x} = (x_1, x_2)$ is presented, the inference mechanism will produce the numerical output

$$y^* = \frac{A_1(x_1)\,B_1(x_2)\,f_1(\mathbf{x}) + A_2(x_1)\,B_2(x_2)\,f_2(\mathbf{x})}{A_1(x_1)\,B_1(x_2) + A_2(x_1)\,B_2(x_2)}$$

A fuzzy-neural network for implementing the above is shown in Figure 7.5. The observed input $\mathbf{x} = (x_1, x_2)$ is presented to Layer 1 by Input Layer 0. The output of Layer 1 is

$$(O_{11}, O_{12}, O_{13}, O_{14}) = (A_1(x_1), A_2(x_1), B_1(x_2), B_2(x_2))$$

where the membership functions A_i, B_i, $i = 1, 2$, are specified in some parametric way from a family of membership functions, such as triangular or Gaussian.

Layer 2 consists of fuzzy neurons with an aggregation operator being some t-norm. We use the product t-norm in this example, in view of the way that product is used in Sugeno-Takagi's inference procedure. The output of Layer 2 is

$$(O_{21}, O_{22}) = (A_1(x_1)\,B_1(x_2), A_2(x_1)\,B_2(x_2))$$

Layer 3 is a normalizer. The output of Layer 3 is

$$\begin{aligned} (O_{31}, O_{32}) &= \left(\frac{O_{21}}{O_{21}+O_{22}}, \frac{O_{22}}{O_{21}+O_{22}}\right) \\ &= \left(\frac{A_1(x_1)B_1(x_2)}{A_1(x_1)B_1(x_2)+A_2(x_1)B_2(x_2)}, \frac{A_2(x_1)B_2(x_2)}{A_1(x_1)B_1(x_2)+A_2(x_1)B_2(x_2)}\right) \end{aligned}$$

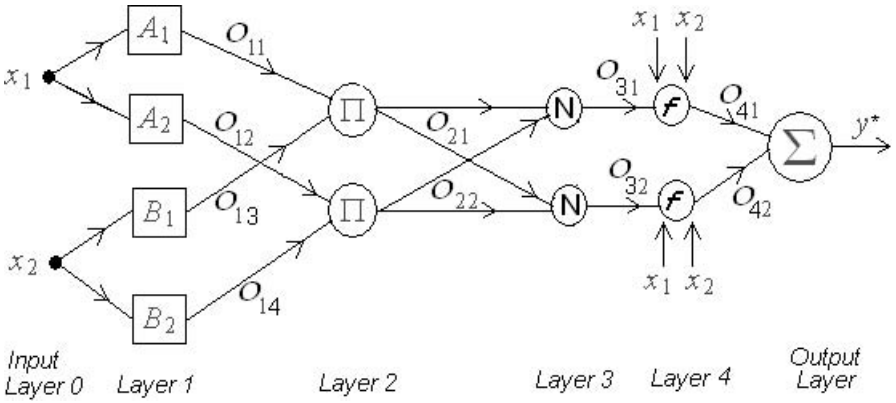


Figure 7.5. First-order Sugeno fuzzy model with two rules

The fuzzy neurons in Layer 4 output the values

$$\begin{aligned}(O_{41}, O_{42}) &= (O_{31}f_1, O_{32}f_2) \\ &= \left(\frac{(A_1(x_1)B_1(x_2))(z_{11}x_1+z_{12}x_2+z_{13})}{A_1(x_1)B_1(x_2)+A_2(x_1)B_2(x_2)}, \frac{(A_2(x_1)B_2(x_2))(z_{21}x_1+z_{22}x_2+z_{23})}{A_1(x_1)B_1(x_2)+A_2(x_1)B_2(x_2)}\right)\end{aligned}$$

Finally, the output layer calculates the control action by summing:

$$y^* = O_{41} + O_{42} = \frac{(A_1(x_1)B_1(x_2))(z_{11}x_1+z_{12}x_2+z_{13})+(A_2(x_1)B_2(x_2))(z_{21}x_1+z_{22}x_2+z_{23})}{A_1(x_1)B_1(x_2)+A_2(x_1)B_2(x_2)}$$

Of course, the above neural network type for representing the inference procedure for a rule base of two rules can be extended in an obvious way to an arbitrary number of rules.

7.3.2 ANFIS learning algorithm

The representation in the preceding section of a neural network is simply a graphical display of the computation steps in the Sugeno-Takagi procedure. In order for this representation to be more useful in implementing the control law, one needs to equip it with an efficient learning algorithm. In conventional neural networks, the backpropagation algorithm is used to learn, or adjust, weights on connecting arrows between neurons from input-output training samples. In the ANFIS structure, the parameters of the premises and consequents play the role of weights. Specifically, when the membership functions A_i^j used in the "If" part of the rules are specified parametrically — that is, the shape is specified and the function is determined by a finite number of parameters, these parameters are called **premise parameters**, whereas the parameters a_i, b_i, c_i, $i = 1, 2$ in the "then" part of the rules are referred to as **consequent parameters**. The ANFIS learning algorithm consists of adjusting the above set of parameters from sample data $\left(\left(x_1^k, x_2^k\right), y^k\right)$, $k = 1, ..., N$.

It is important to keep in mind that when we develop a set of prototype fuzzy rules, we are in fact representing possibly nonlinear input-output relationships. The effectiveness of fuzzy models representing nonlinear input-output relationships depends on the membership functions involved. Thus, the tuning of membership functions is an important issue in fuzzy modeling. This tuning task can be viewed as an optimization problem; neural networks offer a possibility to solve this problem.

In order to train a fuzzy-neural network, we need a set of training data in the form of input-output tuples, and a specification of the rules, including a preliminary definition of the corresponding membership functions. A standard approach is to assume a certain shape for the membership functions so that the membership functions depend on parameters that can be learned by a neural network.

We describe one method for learning the membership functions of the antecedent and consequent parts of fuzzy "If...then..." rules. Suppose an unknown function, or control law, to be realized by a fuzzy inference system is known only through the training set

$$\left\{\left(\mathbf{x}^1, y^1\right), \ldots, \left(\mathbf{x}^K, y^K\right)\right\}$$

where $\mathbf{x}^k = \left(x_1^k, ..., x_n^k\right) \in \mathbb{R}^n$ and $y^k \in \mathbb{R}$. To model the unknown function, we use fuzzy "If...then..." rules R_i, $i = 1, ..., m$, of the following type

$$R_i : \text{If } x_1^k \text{ is } A_i^1 \text{ and ... and } x_n^k \text{ is } A_i^n \text{ then } y = \sum_{j=1}^n z_i^j x_j^k + z_i$$

where A_i^j are fuzzy membership functions and z_i^j are real numbers.

Let O^k be the output from the fuzzy system corresponding to the input $\mathbf{x}^k$. Suppose the fuzzy AND of each rule is implemented by the product, so that the antecedent of the i^{th} rule is given by

$$\alpha_i^k = \prod_{j=1}^n A_i^j\left(x_j^k\right)$$

We could also use other t-norms for modeling the logical connective AND. We compute the output of the system as

$$O^k = \frac{\sum_{i=1}^m \alpha_i^k \left(\sum_{j=1}^n z_i^j x_j^k + z_i^0\right)}{\sum_{i=1}^m \alpha_i^k} = \frac{\sum_{i=1}^m \left(\prod_{j=1}^n A_i^j\left(x_j^k\right)\right)\left(\sum_{j=1}^n z_i^j x_j^k + z_i^0\right)}{\sum_{i=1}^m \prod_{j=1}^n A_i^j\left(x_j^k\right)}$$

and define the measure of error for the k^{th} training pattern as

$$E^k = \frac{1}{2}\left(O^k - y^k\right)^2$$

where O^k is the computed output from the fuzzy system corresponding to the input pattern $\mathbf{x}^k$, and y^k is the desired output, $k = 1, ..., K$. Standard neural network learning methods are used to learn z_i^j, $j = 0, 1, ..., n$ in the consequent part of the fuzzy rule R_i.

Example 7.4 We illustrate the above tuning process by a simplified example. Consider two fuzzy rules, with one input variable x and one output variable y, of the form

$$R_1 : \text{If } x \text{ is } A_1 \text{ then } y = z_1$$
$$R_2 : \text{If } x \text{ is } A_2 \text{ then } y = z_2$$

where the fuzzy sets A_1 and A_2 have sigmoid membership functions defined by

$$\begin{aligned} A_1(x) &= \frac{1}{1+e^{b_1(x-a_1)}} \\ A_2(x) &= \frac{1}{1+e^{b_2(x-a_2)}} \end{aligned}$$

Then a_1, a_2, b_1, and b_2 are the parameter set for the premises, and the antecedent of the rule R_i is simply the value of the membership function $A_i(x)$. The output $O(x)$ of the system, computed by the discrete center-of-gravity defuzzification method, is

$$O(x) = \frac{A_1(x)z_1 + A_2(x)z_2}{A_1(x) + A_2(x)}$$

Given a training set $\left\{\left(x^1, y^1\right), \ldots, \left(x^K, y^K\right)\right\}$, we want to provide the two fuzzy rules with appropriate membership functions and consequent parts to generate the given input-output pairs. That is, we want to learn the parameters a_1, a_2, b_1, and b_2 of the sigmoid membership functions, and the values z_1 and z_2 of the consequent parts.

The measure of error for the k^{th} training pair is defined as

$$E^k = E^k(a_1, b_1, a_2, b_2, z_1, z_2) = \frac{1}{2}\left(O^k(a_1, b_1, a_2, b_2, z_1, z_2) - y^k\right)^2$$

$k = 1, ..., K$, where O^k is the computed output from the fuzzy inference system corresponding to the input pattern x^k, and y^k is the desired output.

The steepest descent method is used to learn z_i in the consequent part of the i^{th} fuzzy rule, and the shape parameters of the membership functions A_1 and A_2. That is,

$$\begin{aligned} z_i(t+1) &= z_i(t) - \eta\frac{\partial E^k}{\partial z_i} = z_i(t) - \eta\left(O^k - y^k\right)\frac{A_i(x^k)}{A_1(x^k) + A_2(x^k)} \\ a_i(t+1) &= a_i(t) - \eta\frac{\partial E^k}{\partial a_i} \\ b_i(t+1) &= b_i(t) - \eta\frac{\partial E^k}{\partial b_i} \end{aligned}$$

where $\eta > 0$ is the learning constant and t indexes the number of adjustments of z_i, a_i, and b_i.

Assuming further that $a_1 = a_2 = a$ and $b_1 = b_2 = b$ simplifies the learning rules because, in this case, the equation $A_1(x) + A_2(x) = 1$ holds for all x from

the domain of A_1 and A_2. The weight adjustments in this case become

$$\begin{aligned} z_i(t+1) &= z_i(t) - \eta \left(O^k - y^k\right) A_i(x^k) \\ a(t+1) &= a(t) - \eta \left(O^k - y^k\right) (z_1 - z_2)\, b A_1(x^k) A_2(x^k) \\ b(t+1) &= b(t) + \eta \frac{\partial E^k(a,b)}{\partial b} \left(O^k - y^k\right) (z_1 - z_2) \left(x^k - a\right) A_1(x^k) A_2(x^k) \end{aligned}$$

as shown in the following computations

$$\begin{aligned} z_i(t+1) &= z_i(t) - \eta \frac{\partial E^k}{\partial z_1} = z_i(t) - \eta \left(O^k - y^k\right) A_i(x^k) \\ a(t+1) &= a(t) - \eta \frac{\partial E^k(a,b)}{\partial a} \\ b(t+1) &= b(t) - \eta \frac{\partial E^k(a,b)}{\partial b} \end{aligned}$$

where

$$\begin{aligned} \frac{\partial E^k(a,b)}{\partial a} &= \left(O^k - y^k\right) \frac{\partial O^k}{\partial a} = \left(O^k - y^k\right) \frac{\partial}{\partial a} \left(z_1 A_1(x^k) + z_2 A_2(x^k)\right) \\ &= \left(O^k - y^k\right) \frac{\partial}{\partial a} \left(z_1 A_1(x^k) + z_2 \left(1 - A_1(x^k)\right)\right) \\ &= \left(O^k - y^k\right) \frac{\partial}{\partial a} \left((z_1 - z_2) A_1(x^k) + z_2\right) \\ &= \left(O^k - y^k\right) \left((z_1 - z_2) \frac{\partial A_1(x^k)}{\partial a}\right) \\ &= \left(O^k - y^k\right) (z_1 - z_2)\, b \frac{e^{b\left(x^k - a\right)}}{\left(1 + e^{b(x-a)}\right)^2} \\ &= \left(O^k - y^k\right) (z_1 - z_2)\, b A_1(x^k) \left(1 - A_1(x^k)\right) \\ &= \left(O^k - y^k\right) (z_1 - z_2)\, b A_1(x^k) A_2(x^k) \end{aligned}$$

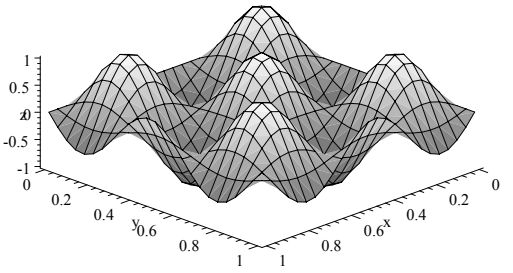
and similarly,

$$\frac{\partial E_k(a,b)}{\partial b} = -\left(O^k - y^k\right) (z_1 - z_2) \left(x^k - a\right) A_1(x^k) A_2(x^k)$$

The following example illustrates how to use the ANFIS algorithm with MATLAB. In practice, the situation is typically like the following. In system identification for indirect neural control, or in direct neural control where we seek to model the control law as a neural network, we are modeling an input-output relationship — that is, approximating some function $y = f(x)$ from the data expressed as a set of fuzzy rules. An appropriate structure of ANFIS is chosen for the approximation problem, guided by a theorem on universal approximation. In simulation studies, various choices of membership functions in fuzzy rules, as well as choices of the number of rules, illustrate the approximation

capability of ANFIS. However, in real-world applications where the control law is unknown, these choices belong to "engineering skill." Because the intended controlled system can be tested, a good approximation can be obtained with time and patience. The point is this: The universal approximation property of ANFIS, as a mathematical theorem, is the theoretical guideline for using ANFIS.

Example 7.5 In this example, we use ANFIS to approximate a function that we know (but pretend not to know). We take for our "unknown" function, $\sin 10x \sin 10y$. The surface determined by this function looks like this:

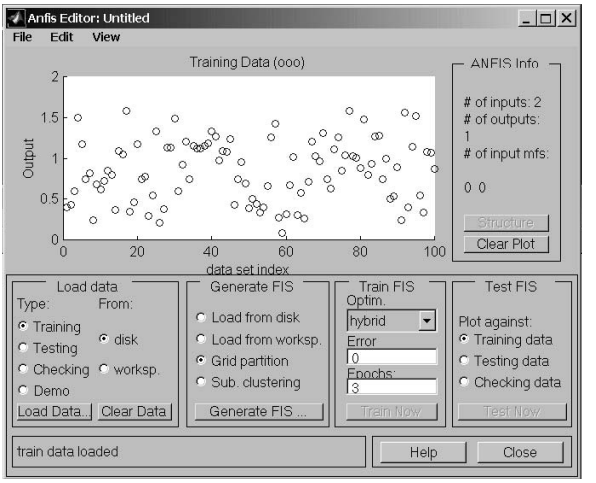


Plot of $\sin 10x \sin 10y$

The training data was obtained from this function by evaluating 100 random pairs (x, y) with $x, y \in [0, 1]$, creating a text file, `sinxsiny.dat`, with three columns of numbers.

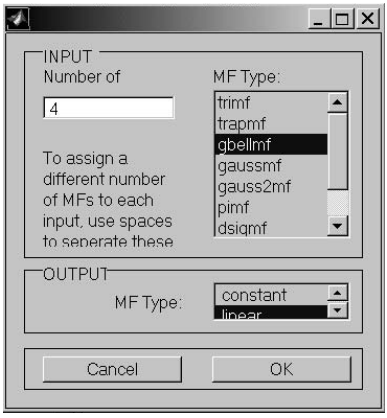
- Open MATLAB, and at the prompt, enter ***anfisedit***

This brings up the following dialog. (The training data box will be empty at this point.)

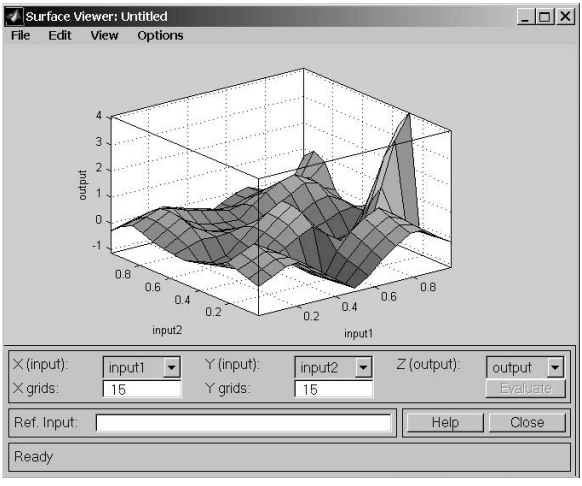


- Choose Load Data; browse and select `sinxsiny.dat`.
- Choose Generate FIS.

This brings up the fuzzy inference system dialog.



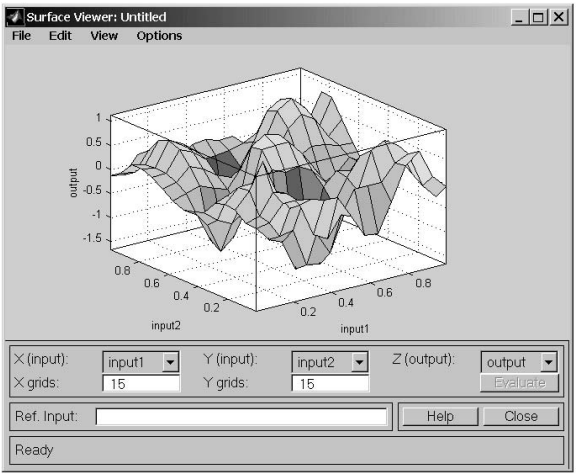
- Under INPUT MF Type, select gbellmf, and set INPUT Number of to 4. Under OUTPUT MF Type, select linear. Choose OK.
- In the Train FIS box, set Error to 0.01 and Epochs to 25, and choose Train Now.
- In the ANFIS Editor, under View, choose View surface to bring up the Surface Viewer with a plot.



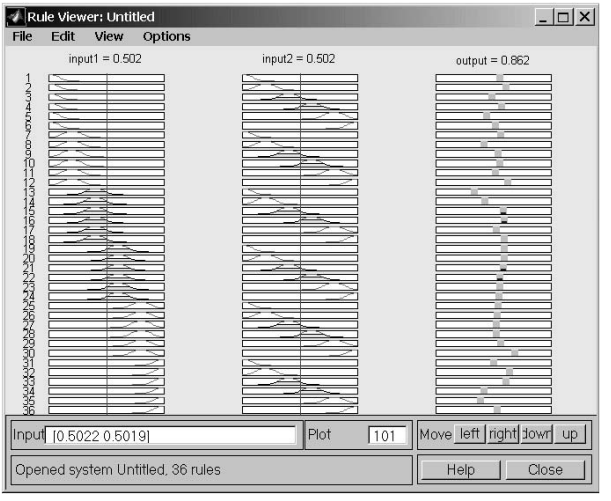
This is not a very good approximation to our function. We can test the effect of increasing the number of membership functions.

- From the ANFIS Editor, choose Generate FIS and set Number of to 6.

The surface will change to something like the following, which is a much better approximation to the function.

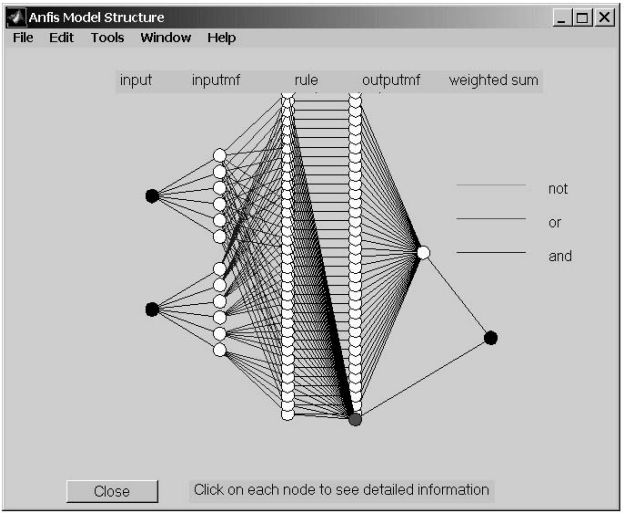


- In the ANFIS Editor, under View, choose View rules to open the Rule Viewer.



The choice of 6 membership functions for each of the two inputs has generated 36 rules.

- In the ANFIS Editor, under ANFIS Info, choose Structure. This shows the structure of the adaptive network.



From the View menu, you can choose Edit FIS properties, Edit membership functions, Edit rules, or Edit anfis. These dialogs provide a number of options for making changes.

7.4 Generating fuzzy rules

Linguistic labels in our natural language convey useful information in human control strategies as well as in other cognitive decision processes. The fuzzy set theory approach to modeling this type of information is based on the thesis that each linguistic label can be represented as a fuzzy subset of an appropriate set U, expressing the semantics of the label. While this seems quite reasonable from a modeling point of view, the concern in applications is determining the membership function of a label. This is related to the more general and more difficult problem of determining rules.

There are several approaches to answer this concern. Rules and membership functions can be given by experts, either in a subjective manner or by using some statistical sampling methods. When experts are not available, but instead, numerical experimental data are at hand, it is possible to use neural networks as a solution to the problem of rule and membership function determination.

With ANFIS, the structure of the rules and the types of the membership functions are specified in advance, and the parameters of the membership functions are learned from the data. However, rules and membership functions can also be determined by using methods that do not presuppose a rule structure. Both the extraction of rules and the determination of membership functions can

be implemented by some kind of clustering. Clustering using neural networks belongs to the domain of unsupervised learning that relies on input data and no corresponding outputs, as opposed to the supervised learning that we have considered so far. Since the problems of membership function determination and of extraction of fuzzy rules from numerical data by neural networks are essentially based on unsupervised learning methods, which are not treated in this text, we elaborate only slightly on these problems in order for the reader to be aware of useful methods and some references.

As in conventional clustering, the goal is to group data points that are similar to each other in some way — that is, forming clusters. Given a set of crisp input-output tuples, or training data $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, ..., n$, fuzzy clustering techniques utilizing neural networks are applied to the input data to determine a collection of fuzzy clusters. Each fuzzy cluster represents one fuzzy "If... then..." rule, where the fuzzy membership functions in the rule are obtained by projecting the cluster to input and output spaces.

We refer readers who are interested in more detail on utilization of neural networks in producing membership functions to a work like Adeli and Hung [2]. See also any of [1, 24, 36, 39, 68].

7.5 Exercises and projects

1. A multi-input/multi-output system is characterized by the following set of nonlinear difference equations:

$$\begin{bmatrix} y_1(k+1) \\ y_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.8 & -0.3 \\ -0.2 & 0.5 \end{bmatrix} \begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} + \begin{bmatrix} -0.1 & -0.2 \\ 0.0 & -0.1 \end{bmatrix} \begin{bmatrix} y_1^2(k-1) \\ y_2^2(k-1) \end{bmatrix} + \begin{bmatrix} 0.7 & 0.1 \\ 0.1 & 0.5 \end{bmatrix} \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix}$$

   The system is stable for inputs $u_1 = u_2 \in [-0.1, 0.1]$.

   (a) Generate a set of data for y_1 and y_2 using uniformly distributed random inputs u_1 and u_2.

   (b) Using the ANFIS editor, generate a trained set of fuzzy inference rules.

   (c) For the inputs $u_1(k) = A\sin(2 * \pi k/5)$ and $u_2(k) = B\cos(2 * \pi k/5)$, test the performance of the trained ANFIS for $A, B \in [-10, 10]$.

   (d) Is ANFIS a suitable approach to identify the nonlinear system? Explain why or why not.

   (e) Discuss the effects of sample size on ANFIS learning.

2. A nonlinear plant is governed by the following difference equation

$$y(k+1) = 0.3y(k) + 0.6\left[y(k-1)\right]^{0.5} + f[r(k)]$$

where $y(k)$ and $r(k)$ are the output and input, respectively, at time step k. The unknown function $f[r(k)]$ has the form $f(r) = 0.5\sin(\pi r)\cos(3\pi r) + 0.3\cos(\pi r)\sin(5r)$.

   (a) Develop a backpropagation neural network to perform system identification of the unknown plant.

   (b) Use ANFIS to identify the unknown system.

   (c) How does the performance of the neural network compare with that of ANFIS?

3. The following dynamical equation describes the behavior of a nonlinear plant

$$y(k+1) = \frac{y(k)u(k)}{1 + \left[|y(k-1)|\right]^{0.3}} - \left[\frac{1 - e^{-u(k)}}{1 + e^{-u(k)}}\right]$$

where $y(k)$ and $u(k)$ are the output and input, respectively, at time step k.

   (a) Assuming the control law $u(k)$ to the system is specified, identify the nonlinear plant dynamics using ANFIS.

   (b) For the desired output of the plant specified by

$$y_d(k) = 0.5\sin(2\pi k)\cos(2\pi k) + 0.3\sin^2(2\pi k/15)$$

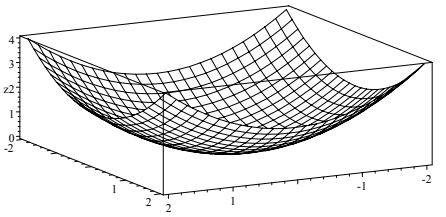
   how well can ANFIS predict the system behavior?

   (c) Compare ANFIS results with that of a neural network-based system identification.

4. Construct an ANFIS that is equivalent to a two-input, two-rule Mamdani fuzzy model. Describe the function that you use to approximate centroid defuzzification. Explain how this function is converted into node functions in the resulting ANFIS.

5. Construct an ANFIS that is equivalent to a two-input, two-rule Larsen fuzzy model. Describe the function that you use to approximate centroid defuzzification. Explain how this function is converted into node functions in the resulting ANFIS.

6. Approximate the function $f(x_1, x_2) = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$ with a Sugeno fuzzy inference system (FIS) using Gaussian antecedent membership functions and linear consequent membership functions.



ANFIS info:

Number of nodes: 21
Number of linear parameters: 12
Number of nonlinear parameters: 8
Total number of parameters: 20
Number of training data pairs: 1681
Number of checking data pairs: 0
Number of fuzzy rules: 4

The four rules:

If x_1 is A_1 and x_2 is B_1 then y is C_1
If x_1 is A_1 and x_2 is B_2 then y is C_2
If x_1 is A_2 and x_2 is B_1 then y is C_3
If x_1 is A_2 and x_2 is B_2 then y is C_4

Chapter 8

APPLICATIONS

In this chapter, we describe some industrial applications of fuzzy control, neural control, neural-fuzzy control, and related neural-fuzzy technology. These examples are intended to give the reader some ideas about the type of problems that are appropriate for using the technology discussed in this text, about how to approach problems that arise in real-world situations, and how to evaluate the performance of the approach.

These examples exemplify the need for soft computing approaches in decision-making. They are intended to demonstrate the applicability of soft computing techniques, their strengths and weaknesses, and the need for evaluating various hybrid approaches. The first two examples treat fuzzy control problems, and the last two examples describe neuro-fuzzy approaches to classification. There is no prescribed methodology or algorithm that dictates how problems can be solved. This provides a motivation to experiment with techniques that allow the combination of known variables in a manner that provides reasonable and realistic outcomes.

8.1 A survey of industrial applications

We indicate briefly here some of industrial applications of fuzzy control, neural control, and neural-fuzzy control. Beginning with Mamdani's application in 1974 to a steam engine, fuzzy control has come to be applied in many areas. In 1987, the Sendai Subway Automatic Train Operations Controller started operating in Sendai, Japan. The fuzzy logic subway control system led to smooth transitions between speeds and a comfortable ride. Today, when you have a particularly smooth ride on a hotel elevator, you might smile and assume that the controller is based on fuzzy logic. Included in industrial applications are those to water purification plants, subway operations, control of temperature, electric current flow, and motions of various machines. Many consumer products utilize fuzzy control, including washing machines, vacuum cleaners, electronic fuel injection systems and automatic cruise control systems of automobiles. There

are many applications to the wide area of pattern recognition, and applications to areas that involve processing of fuzzy information, such as medicine and economics. When fuzzy systems are developed to solve appropriate problems, their typical characteristics include rapid and smooth responses.

Neural networks are more suited than fuzzy logic systems to problems that require a large number of input variables, for example, too many for a rule-based fuzzy system to be feasible. On the other hand, they require large data sets for training. Neural networks are widely used. We mention a few general areas: telecommunication, pattern recognition, quality control, financial and economic forecasting, speech recognition, and many automotive applications.

Neural networks can be used to design membership functions for fuzzy logic systems. Designing appropriate membership functions is at the heart of fuzzy applications. Neural networks can automate the process, saving time and simplifying the tuning process. Many applications are to consumer products such as washing machines, vacuum cleaners, photocopiers, and air conditioning systems. They are also applied in the financial industry. Below, we illustrate two studies of fuzzy control for real-world applications.

8.2 Cooling scheme for laser materials

In this section, we present a fuzzy logic-based approach to the cooling of laser materials developed by N.S. Prasad and N.R. Prasad [58]. The controller design is based on the performance characteristics of commercially available thermoelectric coolers. Simulation results are presented and discussed and the feasibility of implementing such a controller is evaluated.

Solid-state laser materials are susceptible to heating effects caused by optical pumping. The medium basically gets heated up due to the absorption of the pump radiation and non-radiative decay processes. The heating and subsequent cooling gives rise to thermal effects. Thermal gradients result in stresses and strains that produce variations in refractive index, leading to thermo-optic distortions. The distortions lead to self-focusing, depolarization, and other undesired effects that is deleterious to the lasing medium and hence the laser performance. There is extensive on-going research to characterize thermal behavior and to use the results for the design and development of various cooling schemes to minimize detrimental effects [13, 34, 48].

Conventional cooling techniques include passive cooling by surrounding air, liquid cooling, micro-channel cooling, and thermoelectric cooling [12, 38]. Thermoelectric coolers (TECs) are current operated devices that can provide precision cooling. They are easy to implement and operate and have the additional advantages of size and weight. The theory and operation of thermoelectric coolers is widely reported in the literature and they are commercially available [44]. Advances in semiconductor diode-laser pumped solid-state laser systems have necessitated the use of more efficient and miniaturized cooling schemes. As a consequence, TECs are becoming attractive in view of space constraints and operational elegance.

In a practical situation, the thermal modeling of a laser medium is a tedious and cumbersome task. Variations in boundary conditions and pump energy, the presence of surface coatings, presence of material impurities, and so forth, can cause significant deviation from the theoretically predicted results. Since accounting for all such nonlinearities is not possible, alternate methods are needed for efficient heat removal.

Current technology in controlling the operation of thermoelectric devices is restricted to a constant current setting at which optimal performance is expected. If there are changes in the operating environment, the cooling rate has to be manually adjusted to the new operating conditions. Some recent advances in TEC applications have required automatic controllers, which has led to the design and implementation of classical PID type controllers. There is a need for the dynamic control of the cooling process to achieve sustained optimality conditions. It should be noted that over-cooling will cause thermal gradients that severely affect the laser behavior. In this context, a fuzzy logic-based approach provides a promising technique for the control and operation of TECs by providing a wide range of variability. The controller implicitly compensates for highly nonlinear phenomena.

Fuzzy controller design Before proceeding to the design of a controller, it is useful to understand the operating principle of a thermoelectric device. A **thermoelectric device** is a solid-state energy converter that contains arrays of n-type and p-type semiconductors thermally joined in parallel and electrically joined in series at both ends to form a couple. The n-type semiconductor has excess electrons whereas the p-type is electron deficient, so these convert electrical energy to thermal energy and thermal energy to electrical energy, respectively. When a thermoelectric device is converting thermal energy to electrical energy it is called a thermoelectric generator (TEG). When a thermoelectric device is converting electrical energy to thermal energy it is called a thermoelectric cooler (TEC).

TEGs operate on the principle of Seebeck Effect, a solid-state theory that explains current generation through a pair of dissimilar semiconductors due to temperature gradient. Such a heat engine is relatively inefficient and produces power that is suited for systems that have low power requirements. One possible application for such devices is for deep space applications.

TECs, on the other hand, operate on Peltier Effect. As noted above, the n-type and p-type semiconductors are electrically connected in series and thermally connected in parallel between two ceramic plates to form a couple. As current passes through the couples, from the n-type to the p-type, it creates a temperature gradient across the TEC when heat energy is absorbed from the cold junction, transported through the semiconductors by electrons (n-type) and holes (p-type) and transferred to the hot junction. If the direction of current is reversed, heat is transported from the hot junction to the cold junction. The rate of cooling is proportional to the current through the circuit and the number of TEC couples. By appropriately controlling the current input to the TEC,

optimum heat extraction can be achieved. For this purpose, a fuzzy logic-based scheme is proposed. Figure 8.1 illustrates an overview of the control process.

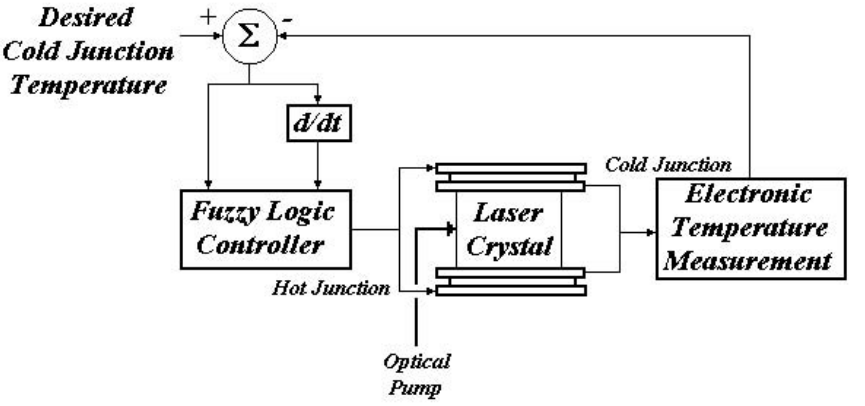


Figure 8.1. Control scheme for thermoelectric cooling of laser materials

The thermal gradients generated by the pumping process in the laser medium are detected by a temperature measurement scheme to provide an input to the fuzzy controller. This temperature normally refers to the peak temperature of the crystal. A temperature error signal is generated by comparing the actual crystal temperature with a reference temperature. This is one of the inputs to the fuzzy controller. For the effective performance of the controller, an additional input, namely, the rate of change in temperature is also required. The output of the controller is the incremental change in current required to regulate the operation of a TEC in order to extract an incremental amount of heat from the surface of the crystal. This process is continued until a quiescent point is achieved where the surface temperature of the crystal is at the desired temperature. For our analysis, a thin slab geometry for an $LiNbO_3$ crystal was assumed. In an ideal case, the one-dimensional temperature profile across the thickness of the slab is parabolic in nature according to the relationship

$$\Delta T = \frac{b^2}{8K_c}Q$$

where, b is the crystal thickness, Q is the heat deposition in watts per cubic meter, K_c is the thermal conductivity of the slab in watts per meter Kelvin, and ΔT is the maximum temperature difference that occurs between the surface and the center of the slab. Without any loss of generality, we assume ΔT is equivalent to the difference in the temperature between the hot and cold junctions of a TEC. Note that these assumptions simplify the model development in order to demonstrate the feasibility of a fuzzy controller implementation. Nevertheless, deviations from ideal conditions are implicitly accounted for in the

fuzzy logic-based system. This feature is the most significant benefit derived from the proposed methodology.

Table 8.1 illustrates the fuzzy associative memory for the inputs and outputs of the fuzzy controller.

Table 8.1. Fuzzy associative memory

		⟵ Temperature Error ⟶						
↑		LNE	MNE	SNE	ZE	SPE	MPE	LPE
Rate of	LNR	LA	LA	LA	LA	LA	SIC	SIC
Change	SNR	LA	LA	LA	LA	SIC	MIC	MIC
in Temp-	ZR	LA	LA	LA	LA	SIC	MIC	LIC
erature	SPR	LA	LA	LA	SIC	MIC	LIC	LIC
↓	LPR	LA	LA	LA	MIC	LIC	LIC	LIC

The fuzzy sets for the temperature error, the rate of change in temperature, and the incremental change in current are defined in Table 8.2.

Table 8.2. Fuzzy set definitions

Temperature Error	Rate of Change in Temperature
LNE = Large Negative Error	LNR = Large Negative Rate
MNE = Medium Negative Error	SNR = Small Negative Rate
SNE = Small Negative Error	ZR = Zero Rate
ZE = Zero Error	SPR = Small Positive Rate
SPE = Small Positive Error	LPR = Large Positive Rate
MPE = Medium Positive Error	
LPE = Large Positive Error	

Incremental Change in TEC Current
LA = Leave Alone
SIC = Small Incremental Change
MIC = Medium Incremental Change
LIC = Large Incremental Change

These subsets are implemented by a set of triangular membership functions, and the centroid method of defuzzification is used. Figure 8.2 illustrates the resulting fuzzy control surface.

Simulation results A MATLAB Simulink model is developed to simulate the TEC heat transfer dynamics and to examine the effectiveness of a fuzzy controller to control a nonlinear process. These simulations are carried out for a lithium niobate crystal of dimensions 1 cm × 1 cm × 3 cm whose thermal conductivity is 5.6 W/ m K. A heat deposition of 5.6 W/ m^3 per unit time interval, a desired surface temperature of 300°K, and initial cold junction temperature of 500°K, are assumed. The time interval chosen depends on the repetition rate of the pumping source and the time constants associated with the cooler and the crystal dynamics. To compute an incremental change in current, an

empirical relationship governing the heat pumped at the cold surface given by a commercially available typical Melcor Brand TEC [44] is considered.

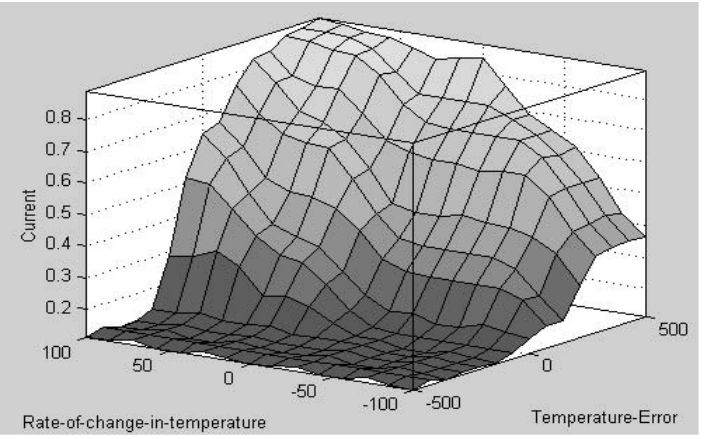


Figure 8.2 Fuzzy control surface

Figure 8.3 displays the schematic of the Simulink model.

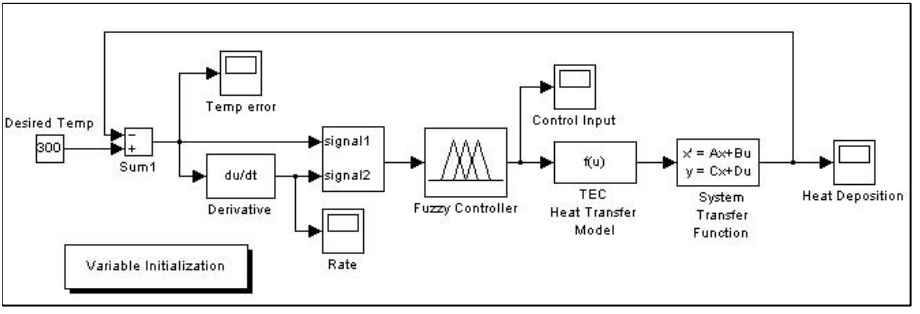


Figure 8.3. Simulink model for TEC heat transfer dynamics

Figures 8.4–8.6 illustrate the controller performance under various operating conditions.

As shown in Figure 8.6, the TEC will achieve the desired temperature in a manner that may be characterized by a "critical" response. However, it is noted that the controller performance can be fine-tuned to yield optimum results in a practical implementation. The primary issue addressed in the simulations above is the effectiveness of a fuzzy controller to achieve the target values in a simple manner without resorting to the design and development of a conventional PID controller. This methodology can be easily extended to include several TECs that may be required to obtain a desired temperature profile across a given surface.

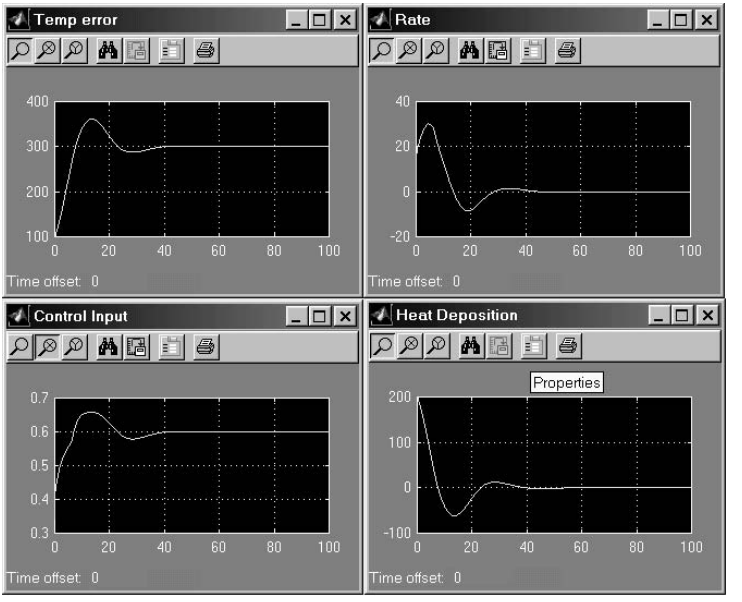


Figure 8.4. Simulation results with prototype membership functions

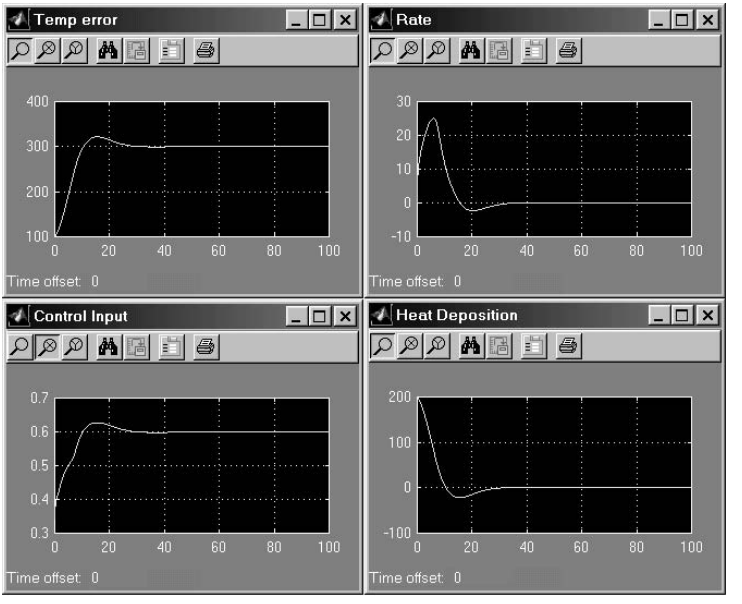


Figure 8.5. Effect of manual fine-tuning of fuzzy control signal

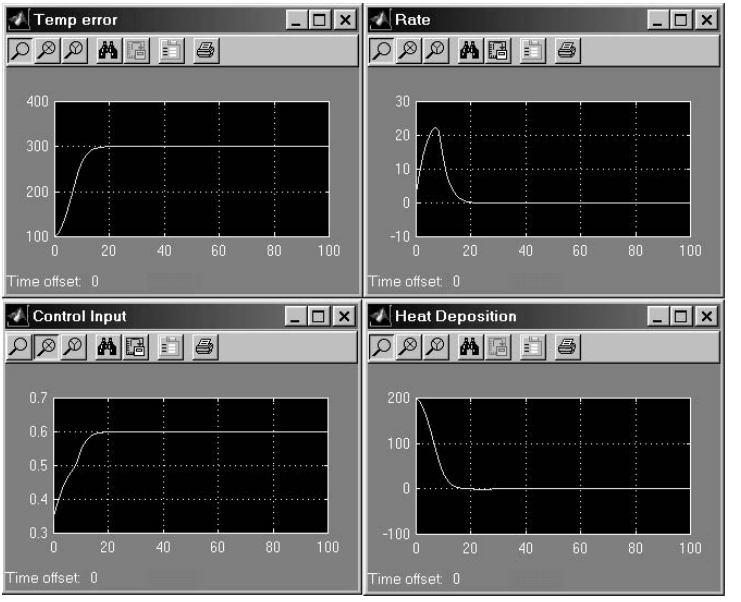


Figure 8.2. Figure 8.6. Critically damped response

8.3 Color quality processing

Color matching is an important issue in many industries such as textiles, automobiles, etc., to maintain uniformity in the color appearance. Although there are analytical methods that provide a means for colorant analysis, their application is cumbersome and involves complex calculations. In this section, we describe a fuzzy logic-based approach for automating the color matching process to obtain a conditional color match between a test sample and a standard based on tristimulus values. A conditional color match implies obtaining a match under a specific set of illuminating and viewing conditions. For a given set of such conditions, computer simulations are performed and discussed. Color evaluation and color mixing for a given material surface are interdependent. The use of a fuzzy logic-based approach yields satisfactory results in terms of color correlation between visual assessment, computed color differences, and colorant composition, and hence can replace the subjective assessments usually performed by humans in the color matching process.

The development of high resolution spectrophotometers and colorimeters, combined with their portability and large data processing abilities, has made the color evaluation process easier and faster. Although these instruments are very useful for rapid pass or fail color inspections in many industries such as the automotive industry and textile industry, the final decision depends primarily upon a subjective visual assessment. Besides spectral analysis, which is useful in colorant selection, the interrelationship between various environmental

factors, metamerism, and texture and composition of the material (substrate), has made visual coordination an acceptable methodology to obtain a repeatable finish and color quality. Subjective assessment in color matching, especially in colors that closely resemble one another, leads to laborious and time-consuming adjustments that have to be performed to obtain the right concentration of the colorants.

In the visual examination of colored objects, the standard — a fully characterized substrate with known colorant concentrations — is compared with a sample — a substrate whose characteristics are unknown, but with the same colorant concentrations as the standard. These are usually placed side by side and viewed at the same time. This visual observation is performed using standardized light sources to arrive at a consensus of whether or not the sample and the standard match in their color representations. If the sample and the standard do not match, the most important issue in this analysis that needs consideration is by how much the sample and the standard deviate from one another. In other words, it is not sufficient to know that the sample and the standard do not match. We need to know the changes in the colorant concentrations to be applied to the sample substrate in order to ultimately match the standard. This gives rise to an area of quantitative judgment in which the eye is less adept than at judging whether two or more objects are identical in their color, and therefore the situation requires the use of measuring instruments such as spectrophotometers for color evaluation [7, 82].

Another important aspect in color evaluation is the issue of **metamerism**. This is a phenomenon in which two or more colored objects viewed under one illuminant may appear to match, while they do not match under another illuminant. This implies that the objects have different reflectance characteristics. Therefore, a pair of objects having different spectral curves but the same color coordinates are called metameric objects. They are said to exhibit metamerism.

Colorant evaluation is based on the computation of tristimulus values [82]. The tristimulus values, generally referred to by the X, Y, and Z coefficients, are based upon the spectral energy and reflectance characteristics of the colorant mixture (a mixture of primary colors) and the substrate (textile, paper, metal, wood, plastic, etc.). The problem in color matching can therefore be stated as follows: Given a standard comprising a colorant mixture of known concentrations, say C_1, C_2, C_3, with tristimulus values X_0, Y_0, and Z_0, the objective is to determine the changes in C_1, C_2, C_3 such that the new colorant mixture when applied to a different substrate sample will result in the desired tristimulus values X_0, Y_0, and Z_0.

Suppose a colorant mixture comprising 30% of C_1, 40% of C_2, and 30% of C_3 when applied to a specific standard substrate produces a reference set of tristimulus values $X_0 = 3.0$, $Y_0 = 4.0$, and $Z_0 = 5.0$. If the same colorant mixture were then applied to a sample fabric, the resulting tristimulus values would be different due to the absorbance and reflectance characteristics of the new substrate. To obtain the same tristimulus values as the reference, the colorant concentrations have to be modified appropriately. Present technology involves the solution of complex mathematical formulas only to obtain an approximate

solution to the changes required in the colorant concentrations. Final analysis is based upon a visual assessment of the color quality that is both time-consuming and prone to errors.

In a fuzzy logic-based approach, a qualitative description of the partial differential equations, governing the change in tristimulus values due to changes in colorant concentration, is coded in the form of a fuzzy associative memory (FAM). Table 8.3 depicts the FAM table for one of the tristimulus coefficients. In this table, one of the two inputs to the FAM is the change in tristimulus value $\Delta X = X_0 - X$, where X_0 is the reference and X is the tristimulus value of the sample, and the other input is the rate of change in tristimulus value with respect to the colorant concentration, namely, $\Delta X/C_1$. The output of the FAM is the change in colorant ΔC_1. Note that the behavior of the other tristimulus values, namely, ΔY and ΔZ bear similar relationships to colorants C_2 and C_3, and consequently the association of subsets in each case will be identical. As such, there are a total of nine FAMs that need to be generated.

Table 8.3. Fuzzy associative memory

				ΔX		
			$\longleftarrow$		$\longrightarrow$	
		LN	SN	ZE	SP	LP
$\uparrow$	LN	SP	SP	ZE	SN	SN
$\Delta X/C_1$	SN	LP	SP	ZE	SN	SN
$\downarrow$	ZE	LN	LN	ZE	LP	LP
	SP	SN	SN	ZE	SP	LP
	LP	SN	SN	ZE	SP	SP

LN = Large Negative
SN = Small Negative
ZE = Zero
SP = Small Positive
LP = Large Positive

In the preceding table, the fuzzy sets representing the input and output may be defined in terms of triangular, trapezoidal, Gaussian, or any other suitable membership functions. Of particular significance is the column of entries pertaining to the change in tristimulus value $\Delta X = ZE$ subset. Note that in a real-time dynamical process, one needs to consider the appropriate control actions to compensate for the degree of variation in the "velocity" term even when the "position" term is part of the zero error subset. This is necessary to prevent a deadband in the output control function and to maintain a stable response. However, in the color matching problem when the tristimulus values are in the zero subset, because the process is not a real-time system, the rate of change in tristimulus value with respect to a colorant has no effect on the change in colorant. This is so because the process is static and a rate of change in tristimulus value with respect to a colorant cannot be associated. This would imply that no change in colorant is required when the deviation between the tristimulus value of the sample and that of the standard is part of the zero error subset. Figure 8.7 illustrates the subsets in the form of triangular membership functions.

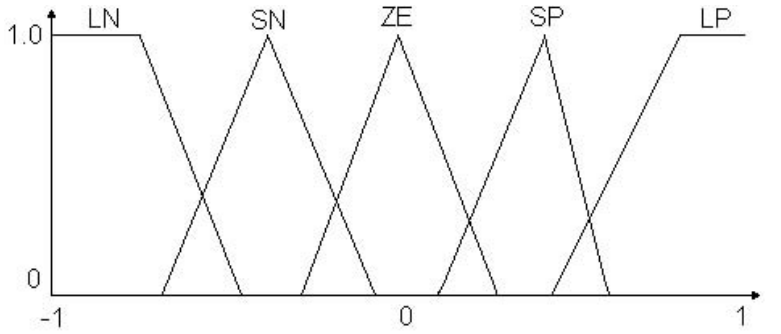


Figure 8.7 Triangular membership functions defining the inputs and the output

The fuzzy control surface for the set associations described in the preceding table is shown in Figure 8.8.

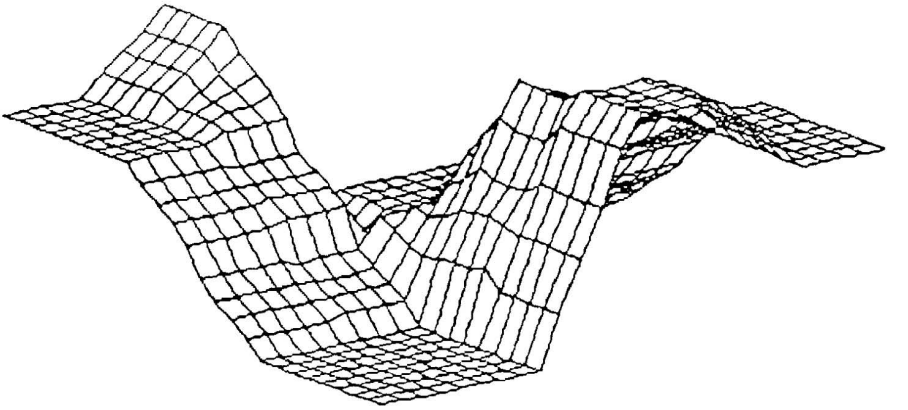


Figure 8.8 Fuzzy control surface using triangular membership functions

Figure 8.9 illustrates the overall schematic of the fuzzy logic-based color matching scheme. Consider, for example, the outputs corresponding to the FAMs marked 1, 2, and 3. Note that the output of each FAM is the change in colorant AC_1. It is natural, in this case, to select the minimum value of the three outputs to reduce the possibility of overshoot in the net colorant concentration. As such, the minimum value of the outputs from each set of FAMs in Figure 8.9 are used to update the colorant concentrations. This update is computed by adding the change in colorant concentration to the previously computed value of colorant concentration. These are shown in Figure 8.9 by the summing junctions where $C_1 = C_1 + \Delta C_1$, $C_2 = C_2 + \Delta C_2$, and $C_3 = C_3 + \Delta C_3$.

In an experimental setup, after computing the colorant concentrations, the colorants are mixed and applied to the appropriate sample/substrate and a spectrophotometric analysis is conducted. The resulting reflectance curves provide a basis for determining the new tristimulus values, namely, X, Y, and Z. Note that in the spectrophotometric analysis, all nonlinearities attributable to the

absorption characteristics of the sample, sample thickness, and sample texture are implicitly accounted in the reflectance characteristics of the sample.

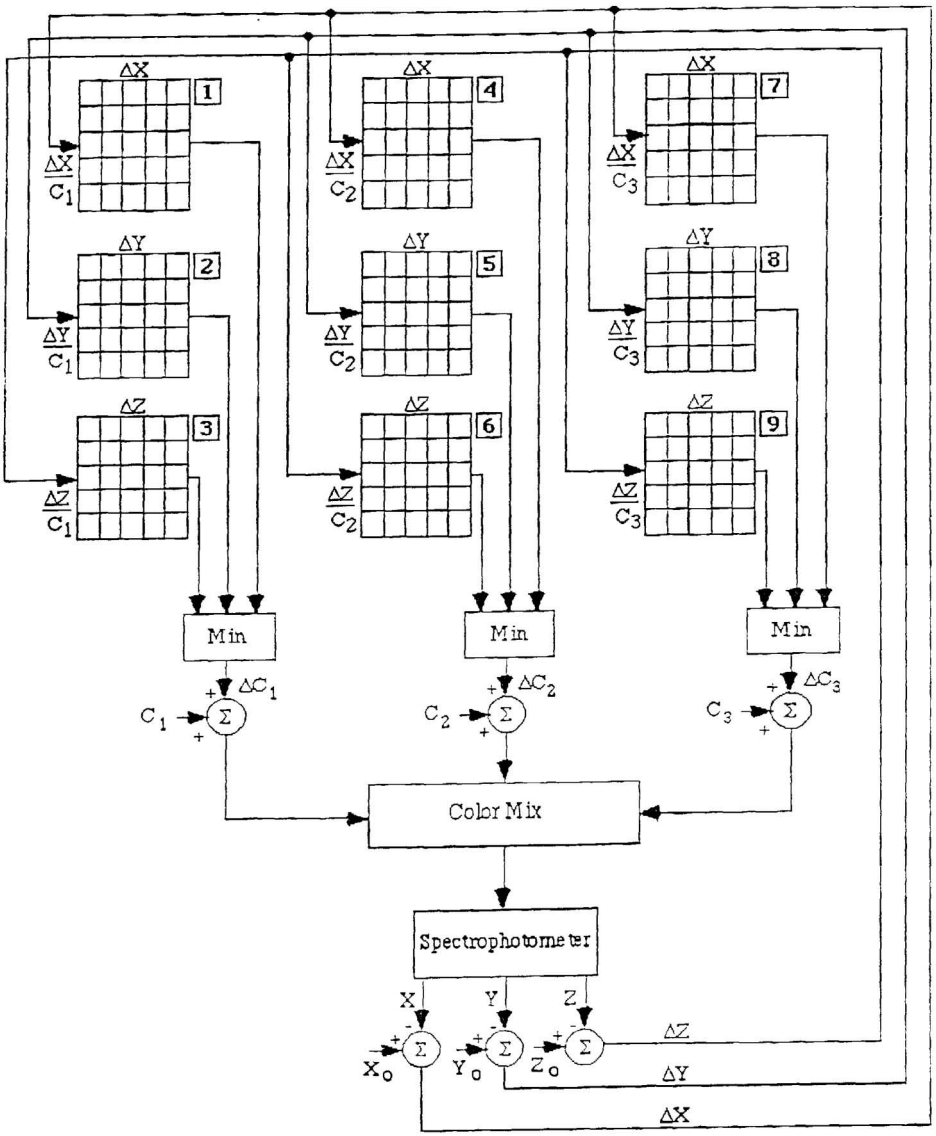


Figure 8.9. A fuzzy logic-based control scheme for colorant mixing

However, in a simulation environment, the mathematical equations governing the determination of the new tristimulus values X, Y, and Z are required.

These equations, given in [7], are:

$$\begin{aligned} X - X_0 &= (X - X_1)\Delta C_1 + (X - X_2)\Delta C_2 + (X - X_3)\Delta C_3 \\ Y - Y_0 &= (Y - Y_1)\Delta C_1 + (Y - Y_2)\Delta C_2 + (Y - Y_3)\Delta C_3 \\ Z - Z_0 &= (Z - Z_1)\Delta C_1 + (Z - Z_2)\Delta C_2 + (Z - Z_3)\Delta C_3 \end{aligned} \tag{8.1}$$

Equations 8.1 can be rewritten as

$$\begin{aligned} X &= \frac{X_0 - X_1\Delta C_1 - X_2\Delta C_2 - X_3\Delta C_3}{1 - \Delta C_1 - \Delta C_2 - \Delta C_3} \\ X &= \frac{Y_0 - Y_1\Delta C_1 - Y_2\Delta C_2 - Y_3\Delta C_3}{1 - \Delta C_1 - \Delta C_2 - \Delta C_3} \\ Z &= \frac{Z_0 - Z_1\Delta C_1 - Z_2\Delta C_2 - Z_3\Delta C_3}{1 - \Delta C_1 - \Delta C_2 - \Delta C_3} \end{aligned} \tag{8.2}$$

The evaluation of Equations 8.2 to obtain the tristimulus values of the sample require a set of nine look-up tables, namely, (X_1 versus C_1), (X_1 versus C_2), (X_1 versus C_3), ..., (Z_3 versus C_3) that have to be experimentally determined for a given sample. The tristimulus values X, Y, and Z of the sample are then compared with that of the standard, namely, X_0, Y_0, and Z_0. If the difference in tristimulus values between the standard and that of the sample is within the acceptable tolerance, the colorant concentrations will result in a match between the standard and the sample. If not, a new change in colorant concentration is computed and added to the previously computed values of C_1, C_2, and C_3.

Simulation results Figures 8.10 and 8.11 illustrate simulation results for a case where the initial colorants C_1, C_2, and C_3, and the tristimulus values X_0, Y_0, and Z_0 are specified.

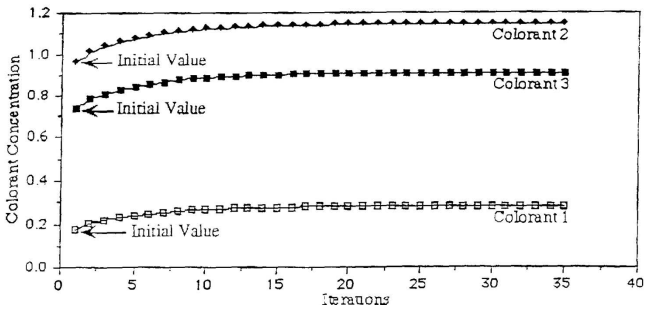


Figure 8.10. Convergence of colorant concentrations to yield desired X_0, Y_0, and Z_0

For the purpose of simulation, values of the colorants that are smaller than those of the standard are chosen initially and their tristimulus values are determined as a representative description of the sample. It is clear from the results that the colorant concentrations are incrementally adjusted following each iteration to yield ultimately the desired tristimulus values of the standard. It should

be pointed out that in dye-mixing processes, the desired color match is obtained by systematically adding the colorants until the desired color is obtained. Since adjustments in color cannot be made by removing the colorant once it has been applied to a sample, the initial concentrations of the colorant need to be less than those specified by the standard to assure an additive mixing process.

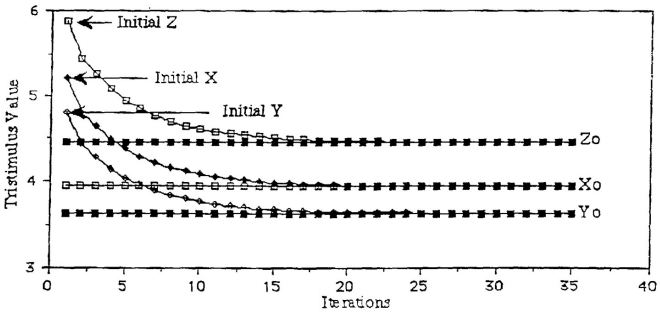


Figure 8.11. Convergence of the tristimulus values of the sample to the desired X_0, Y_0, and Z_0

Another important issue in dye mixing processes is that other additives are required for pH control, mixture consistency, etc., and to develop proper bonding characteristics between the colorant mixture and the sample substrate. To include the effect of these additives, an additional set of FAMs similar to that shown in Table 8.3 should be incorporated into the schematic shown in Figure 8.9.

In the simulation study, no attempts were made to optimize the performance of the fuzzy controller to yield a "critical" response in the color matching process. This can be achieved by adjusting the membership functions appropriately to produce a faster convergence and thereby minimizing the number of samples that need to be examined under a spectrophotometer.

The proposed methodology can be easily implemented using commercially available fuzzy rule chips that include appropriate AID and DIA converters and other associated electronic components. The fuzzy logic hardware, the dye-mixing process, and spectrophotometer and associated accessories can all be integrated into a single color mixing and matching unit.

8.4 Identification of trash in cotton

In order to maintain uniform cotton quality, the cotton industry adheres to standards for cotton classification based on several quality factors. The presence of any extraneous material adversely affects some of these quality factors. Mechanically harvested cotton contains large quantities of trash in addition to useful cotton fiber. The ginning process separates cotton fiber from trash material, and various cleaning machines are used to remove the trash. Classification of trash objects at critical locations in the ginning process can be used for the

dynamic allocation of cleaning equipment to produce high-quality cotton. In this example, we describe a computer-vision-based system for on-line identification of trash types. A fuzzy inference system (FIS), namely ANFIS, is used to classify trash types.

For the purposes of this study, trash is classified into four types: bark, stick, leaf, or pepper trash. Each training sample contained a single type of trash, and two sets of training samples were prepared for each type of trash. Trash objects were chosen from a bale of cotton to include all possible extremes with regards to size and shape of the trash types. The features used to provide distinguishing characteristics among the trash types were

- *area* (the number of pixels within the feature)
- *solidity* (area/convex area)
- E^{dif} (the difference between extent measures at 0° and 45°, where extent = Net Area/Bounding Rectangle)

These shape descriptors are numerical values and for any classifier to perform with a high degree of classification accuracy, it is required that these numerical values are distinct among the trash types. This requires that the trash objects commonly found in ginned cotton have forms or shapes that prescribe to the definition of a typical trash type. These definitions can be easily expressed based on human perception and judgment. This is based on the experience and knowledge humans have acquired over a long period of time. Based on certain physical attributes associated with each trash type, it is possible to describe a certain type of trash object as bark, stick, leaf, or pepper.

In this section, an adaptive network is described that is functionally equivalent to a fuzzy inference system used for the classification of the trash types. The architecture that is used in the development of the classifier is referred to as an adaptive network-based fuzzy inference system (ANFIS).

To understand better the performance of the structure, the ANFIS architecture is presented for a system with two inputs and a single output. The principle is expanded in developing a system to classify the trash types using *area*, *solidity,* and E^{dif} as inputs to the structure. Fuzzy rules developed for the structure are also presented and the classification results for the five test samples are examined to evaluate the performance of the ANFIS as a classifier.

ANFIS architecture Consider a fuzzy inference system that has two inputs x and y and a singleton z as its output. For a first-order Sugeno model, a common rule set with two fuzzy "If...then..." rules is as follows:

$$\begin{aligned} &\text{Rule 1: If } x \text{ is } A_1 \text{ and } y \text{ is } B_1 \text{ then } z = f_1 = a_0^1 + a_1^1 x + a_2^1 y \\ &\text{Rule 2: If } x \text{ is } A_2 \text{ and } y \text{ is } B_2 \text{ then } z = f_2 = a_0^2 + a_1^2 x + a_2^2 y \end{aligned} \tag{8.3}$$

Figure 8.12 (a) shows the reasoning mechanism for this Sugeno model. The corresponding equivalent ANFIS architecture, which we discuss now, is shown

in Figure 8.12 (b), where nodes of the same layer have similar functions. In the discussion, the term $O_{l,i}$ denotes the output of the i^{th} node in layer l.

Layer 1: Every node i in this layer is an adaptive node with a node function

$$\begin{array}{ll} O_{1,i} = A_i(x), & \text{for } i = 1,\ 2 \\ O_{1,i} = B_{i-2}(x), & \text{for } i = 3,\ 4 \end{array} \tag{8.4}$$

where x (or y) is the input to node i and A_i (or B_{i-2}) is a linguistic label (such as "small" or "large") associated with the node. In other words, $O_{1,i}$ is the membership grade of a fuzzy set

$$A = (A_1,\ A_2,\ B_1\ or\ B_2)$$

and it specifies the degree to which the given input x (or y) satisfies a quantifier A. Here the membership function for A can be an appropriately parameterized membership function such as the generalized bell function

$$A(x) = \frac{1}{1 + \left|\frac{x-c_i}{a_i}\right|^{2b}}$$

where $\{a_i,\ b_i,\ c_i\}$ is the parameter set. As the values of these parameters change, the bell-shaped function varies accordingly, thus exhibiting various forms of membership functions for fuzzy set A. Parameters in this layer are generally referred to as **premise parameters**.

Layer 2: Every node in this layer is a fixed node labeled π, whose output is the product of all the incoming signals:

$$O_{2,i} = w_i = A_i(x)B_i(y), \qquad \text{for } i = 1,\ 2. \tag{8.5}$$

Each node output represents the firing strength of a rule. In general, any other t-norm operators, all of which perform a fuzzy AND, can be used as the node function in this layer.

Layer 3: Every node in this layer is a fixed node labeled N. The i^{th} node calculates the ratio of the i^{th} rule's firing strength to the sum of all rule's firing strengths:

$$O_{3,\,i} = \widehat{w}_i = \frac{w_i}{w_1 + w_2} \qquad \text{for } i = 1,\ 2 \tag{8.6}$$

The outputs of this layer are referred to as **normalized firing strengths**.

Layer 4: Every node i in this layer is an adaptive node with a node function

$$O_{4,\,i} = \widehat{w}_i f_i = \widehat{w}_i(a_0^i + a_1^i x + a_2^i y) \qquad \text{for } i = 1,\ 2 \tag{8.7}$$

where $\widehat{w}_i$ is the normalized firing strength from layer 3 and $\left\{a_0^i,\ a_1^i,\ a_2^i\right\}$ is the parameter set of this node. Parameters in this layer are referred to as **consequent parameters**.

Layer 5: The single node in this layer is a fixed node labeled $\sum$, which computes the overall output as the summation of all incoming signals:

$$\text{overall output } = O_5 = \sum_i \widehat{w}_i f_i \ = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

The constructed adaptive network is functionally equivalent to a Sugeno fuzzy model. It is important to note that the structure of this adaptive network is not unique. For instance, Layers 3 and 4 can be combined to obtain an equivalent network with only 4 layers. By the same notion, the weight normalization can be performed in the last layer.

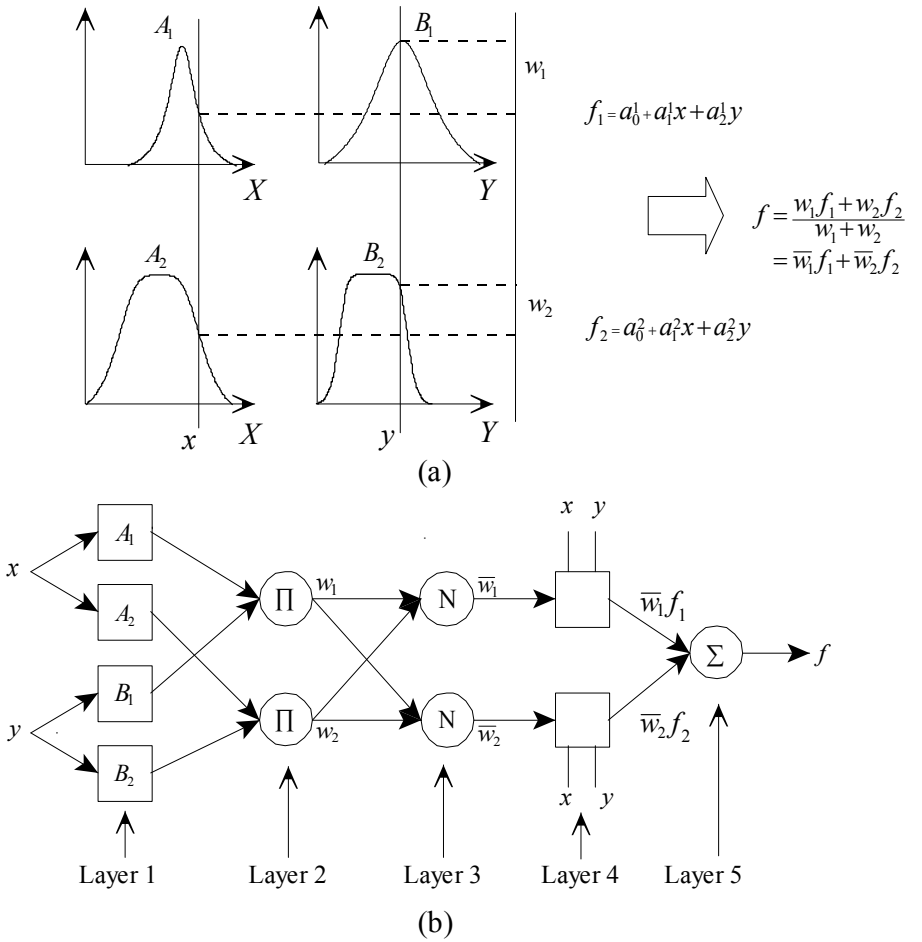


Figure 8.12. (a) A two-input first-order Sugeno model with two rules (b) The equivalent ANFIS architecture

The extension from the Sugeno ANFIS to Tsukamoto ANFIS is straightforward, where the output for each rule (f_i, $i = 1, 2$) is induced jointly by a

consequent membership function and a firing strength. For the Mamdani fuzzy inference system with max-min composition, a corresponding ANFIS can be constructed if discrete approximations are used to replace the integrals in the centroid defuzzification scheme. In this example, only the ANFIS architecture for the first-order Sugeno fuzzy model is used due to its transparency and efficiency. However, detailed discussion on the other ANFIS architecture can be found in [36].

Hybrid-learning algorithm From the ANFIS architecture in Figure 8.12 (a), we observe that when the values of the premise parameters are fixed, the overall output can be expressed as a linear combination of the consequent parameters. The output f in Figure 8.12 (b) can be written as

$$\begin{aligned} f &= \frac{w_1}{w_1+w_2}f_1 + \frac{w_2}{w_1+w_2}f_2 \\ &= \widehat{w}_1(a_0^1 + a_1^1 x + a_2^1 y) + \widehat{w}_2(a_0^2 + a_1^2 x + a_2^2 y) \\ &= (\widehat{w}_1)a_0^1 + (\widehat{w}_1 x)a_1^1 + (\widehat{w}_1 y)a_2^1 + (\widehat{w}_2)a_0^2 + (\widehat{w}_2 x)a_1^2 + (\widehat{w}_2 y)a_2^2 \end{aligned} \tag{8.8}$$

which is linear in the consequent parameters $a_0^1,\ a_1^1,\ a_2^1,\ a_0^2,\ a_1^2,\ a_2^2$.

The consequent parameters can be obtained by solving the following over-constrained, simultaneous equations

$$\begin{bmatrix} \widehat{w}_1^{(1)} & \widehat{w}_1^{(1)}x^{(1)} & \widehat{w}_1^{(1)}y^{(1)} & \widehat{w}_2^{(1)} & \widehat{w}_2^{(1)}x^{(1)} & \widehat{w}_2^{(1)}y^{(1)} \\ \widehat{w}_1^{(2)} & \widehat{w}_1^{(2)}x^{(2)} & \widehat{w}_1^{(2)}y^{(2)} & \widehat{w}_2^{(2)} & \widehat{w}_2^{(2)}x^{(2)} & \widehat{w}_2^{(2)}y^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \widehat{w}_1^{(n)} & \widehat{w}_1^{(n)}x^{(n)} & \widehat{w}_1^{(n)}y^{(n)} & \widehat{w}_2^{(n)} & \widehat{w}_2^{(n)}x^{(n)} & \widehat{w}_2^{(n)}y^{(n)} \end{bmatrix} \begin{bmatrix} a_0^1 \\ a_1^1 \\ a_2^1 \\ a_0^2 \\ a_1^2 \\ a_2^2 \end{bmatrix} = \begin{bmatrix} d^{(1)} \\ d^{(2)} \\ \vdots \\ d^{(n)} \end{bmatrix} \tag{8.9}$$

where $\left[\left(x^{(k)},\ y^{(k)}\right), d^{(k)}\right]$ are the k^{th} training pair $k = 1,\ 2,\ \cdots,\ n$, and $\widehat{w}_1^{(k)}$ and $\widehat{w}_2^{(k)}$ are the outputs of Layer 3 associated with the inputs $\left(x^{(k)},\ y^{(k)}\right)$.

Equation 8.9 can be expressed in matrix-vector form as

$$\mathbf{A}\mathbf{x} = \mathbf{d} \tag{8.10}$$

where

$$\mathbf{x} = [a_0^1,\ a_1^1,\ a_2^1,\ a_0^2,\ a_1^2,\ a_2^2]^T, \quad \mathbf{d} = [d^{(1)},\ d^{(1)},\ \cdots, d^{(n)}]^T$$

and $\mathbf{A}$ is a matrix formed by the elements $\widehat{w}_1^{(k)},\ \widehat{w}_2^{(k)},\ x^{(k)},\ y^{(k)}$. There are several approaches to solve these kinds of constrained equations that can be used to obtain the consequent parameters. One of the most concise ways to solve Equation 8.10 if $\mathbf{A}^T\mathbf{A}$ is nonsingular is to use the pseudoinverse technique

$$\mathbf{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{d} \tag{8.11}$$

In many instances, the row vectors of matrix $\mathbf{A}$ (and the corresponding elements in $\mathbf{d}$) are obtained sequentially; hence, it is desirable to compute the least squares estimate of $\mathbf{x}$ in Equation 8.10 recursively. Let the i^{th} row vector of the matrix $\mathbf{A}$ defined in Equation 8.10 be $\mathbf{a}_i$ and the i^{th} element of $\mathbf{d}$ be $d^{(i)}$; then $\mathbf{x}^*$ can be calculated recursively using the formulas

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{x}_i + \mathbf{S}_{i+1}\mathbf{a}_{i+1}^T\left(d^{(i+1)} - \mathbf{a}_{i+1}\mathbf{x}_i\right) \\ \mathbf{S}_{i+1} &= \mathbf{S}_i - \frac{\mathbf{S}_i\mathbf{a}_{i+1}^T\mathbf{a}_{i+1}\mathbf{S}_i}{1+\mathbf{a}_{i+1}\mathbf{S}_i\mathbf{a}_{i+1}^T}, \qquad i = 0,\ 1,\ \cdots,\ n-1 \\ \mathbf{x}^* &= \mathbf{x}_n \end{aligned} \tag{8.12}$$

with the initial conditions of

$$\begin{aligned} \mathbf{x}_0 &= \mathbf{0} \\ \mathbf{S}_0 &= \gamma \cdot \mathbf{I} \end{aligned}$$

where γ is a positive large number and $\mathbf{I}$ is the identity matrix.

The least squares estimate of $\mathbf{x}$ in Equations 8.12 can also be interpreted as a Kalman filter for the process

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{x}(k) \\ \mathbf{y}(k) &= \mathbf{A}(k)\mathbf{x}(k) + \text{ noise} \end{aligned} \tag{8.13}$$

where $\mathbf{x}(k) \equiv \mathbf{x}_k$, $\mathbf{y}(k) \equiv d^{(k)}$, and $\mathbf{A}(k) = \mathbf{a}_k$. Therefore, the formulas in Equation 8.13 are usually referred to as a Kalman filter algorithm.

In the forward pass of the hybrid learning algorithm, node outputs go forward until Layer 4 and the consequent parameters are identified by the least squares method outlined above. In the backward pass, the signals that propagate backwards are the error signals and the premise parameters are updated by the gradient descent method. Table 8.4 summarizes the various activities during each pass [36].

Table 8.4. Parameter update during the forward and backward passes in the hybrid-learning procedure for ANFIS.

Signal flow direction	Forward pass	Backward pass
Consequent parameters	Least-squares estimator	Fixed
Premise parameters	Fixed	Gradient descent method
Signals	Node outputs	Error signals

ANFIS classification results The ANFIS algorithm was used to identify the trash types, and its performance was evaluated as a classifier. The classification results were compared with the results obtained from fuzzy clustering and backpropagation neural network algorithms. The inputs to the network are *area*, *solidity,* and E^{dif} measures for the trash objects. These inputs form the linguistic variables for the fuzzy "If...then..." rules and are assigned the linguistic values as shown in Table 8.5.

Table 8.5. Linguistic variables and linguistic values for fuzzy "If... then..." rules

Linguistic Variable	Linguistic Values
Area	{small, large}
Solidity	{small, medium, large}
E^{dif}	{small, large}

Formulation of membership functions Even though the training set used to develop the "If... then..." rules is a set of number pairs, the membership functions normally used to implement these rules have a universe of discourse X consisting of the real line $\mathbb{R}$. These membership functions are chosen from known parameterized families of functions such as the generalized bell functions.

A generalized bell membership function, commonly referred to as bell MF, is characterized by three parameters namely a, b, c.

$$\text{bell}(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}$$

A desired, generalized bell membership function can be obtained with the proper selection of the parameters a, b, c. The parameters a and c represent the width and the center of the bell function, and b represents the slopes at the crossover points. Figure 8.13 illustrates the parameters that describe the generalized bell function.

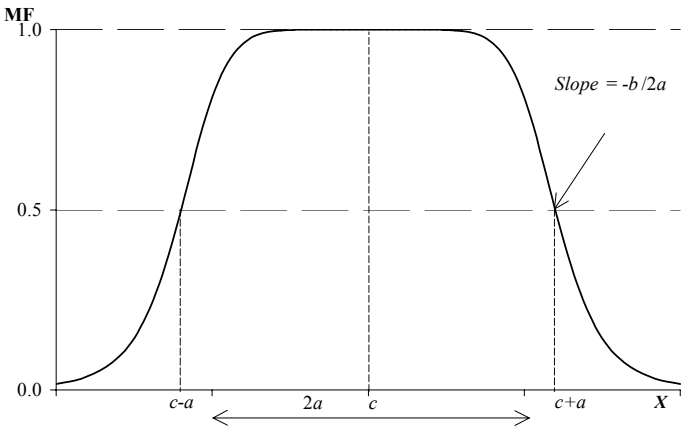


Figure 8.13. Parameter description of a generalized bell function

Various other membership functions such as triangular, trapezoidal, Gaussian, and sigmoidal can be used in the formulation of membership functions. See Section 3.2 for details. The triangular and trapezoidal membership functions, due to their simplicity and computational efficiency, are used extensively in the formulation of membership functions for many applications. However, these membership functions consist of line segments and are not smooth at the corner specified by the parameters. The Gaussian, the generalized bell function, and

the sigmoidal membership functions are smooth and nonlinear functions and are increasingly popular for specifying fuzzy sets. The generalized bell function has one more parameter than the Gaussian membership functions, resulting in an extra degree of freedom to adjust the steepness at the crossover points [36]. This is one of the reasons the generalized bell function is used in the formulation of the membership functions in this example.

Fuzzy partition of input space Since *area* measures for pepper objects are small compared to the other types of trash, it is assigned the linguistic values *small* and *large*. *Solidity* measures for bark objects are typically small compared to the measures for leaf and stick objects. However, for some bark objects, this value can be slightly higher than the typical values for ideal bark objects. Hence, the linguistic variable *solidity* is assigned the values *small*, *medium*, and *large*. Since E^{dif} measure for bark and stick objects are the highest, while the measures for leaf and pepper objects are small, it is assigned the values of *small* and *large* to distinguish the stick objects from the other trash types. Table 8.5 is a summary of the linguistic variables and the linguistic values used to obtain the premise and consequent parameters of the ANFIS structure. Figure 8.14 illustrates the ANFIS structure developed for the identification of the trash types [63, 64, 65, 66].

Fuzzy rules for trash identification The linguistic variables *area*, *solidity,* and E^{dif} with the linguistic values assigned in Table 8.5 are used to develop the Sugeno fuzzy rules for the various trash types. The fuzzy rules for describing the four trash types are as follows:

R^1: If *area* is small and *solidity* is small and E^{dif} is small, then trash type is pepper.

R^2: If *area* is small and *solidity* is small and E^{dif} is large, then trash type is pepper.

R^3: If *area* is small and *solidity* is medium and E^{dif} is small, then trash type is pepper.

R^4: If *area* is small and *solidity* is medium and E^{dif} is large, then trash type is pepper.

R^5: If *area* is small and *solidity* is large and E^{dif} is small, then trash type is pepper.

R^6: If *area* is small and *solidity* is large and E^{dif} is large, then trash type is pepper.

R^7: If *area* is large and *solidity* is small and E^{dif} is small, then trash type is bark.

R^8: If *area* is large and *solidity* is small and E^{dif} is large, then trash type is bark.

R^9: If *area* is large and *solidity* is medium and E^{dif} is small, then trash type is bark.

R^{10}: If *area* is large and *solidity* is medium and E^{dif} is large, then trash type is stick.

R^{11}: If *area* is large and *solidity* is large and E^{dif} is small, then trash type is leaf.

R^{12}: If *area* is large and *solidity* is large and E^{dif} is large, then trash type is stick.

Generally, the number of rules for any system is the product of the number of linguistic values of all the linguistic variables. Since we have two linguistic values {small, large} for *area* and E^{dif} and three linguistic values {small, medium, large} for *solidity,* we have a total of 12 fuzzy rules. The 12 rules that describe the trash types are summarized in Table 8.6.

Table 8.6. TSK fuzzy rules for trash identification with 232-partition[1]

	IF	AND	AND	THEN
Rule	*Area* is	*Solidity* is	E^{dif} is	*Trash Type* is
R^1	Small	Small	Small	Pepper
R^2	Small	Small	Large	Pepper
R^3	Small	Medium	Small	Pepper
R^4	Small	Medium	Large	Pepper
R^5	Small	Large	Small	Pepper
R^6	Small	Large	Large	Pepper
R^7	Large	Small	Small	Bark
R^8	Large	Small	Large	Bark
R^9	Large	Medium	Small	Bark
R^{10}	Large	Medium	Large	Stick
R^{11}	Large	Large	Small	Leaf
R^{12}	Large	Large	Large	Stick

Based on these rules, the ANFIS is trained to modify the premise parameters and the consequent parameters for a given pair of input-output training pairs. The outputs at Layer 4 of the ANFIS structure are

$$f^i = a_0^i + a_1^i x_1 + a_2^i x_2 + a_3^i x_3 \qquad i = 1, \cdots, 12$$

[1]The membership partition of the input variable *area*, *solidity,* and E^{dif} are {(Small, Large), (Small, Medium, Large), (Small, Large)} and is represented as 232-partition in the text.

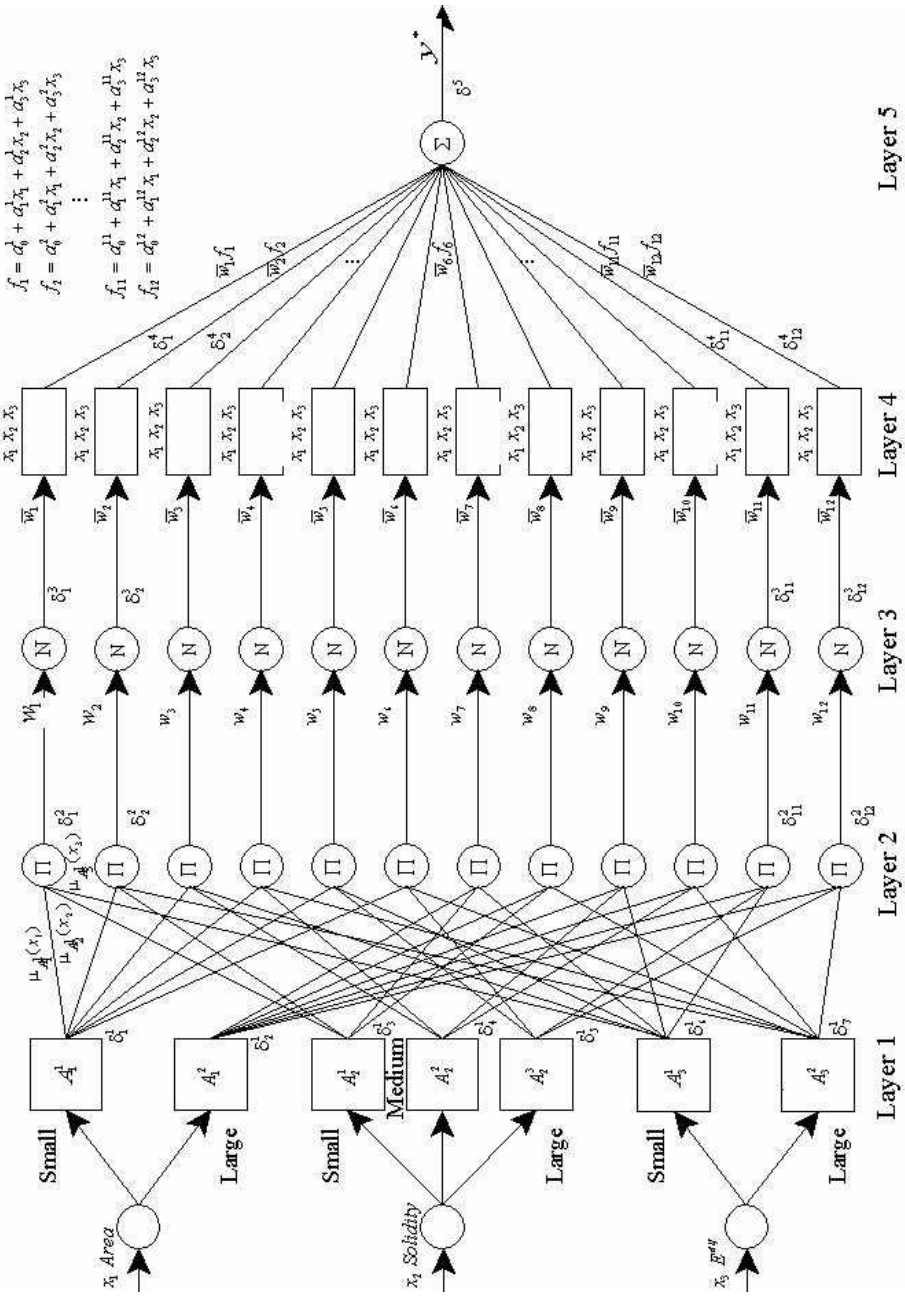


Figure 8.14. ANFIS structure for identification of trash types

There are a total of 69 parameters that are updated during one complete forward and backward pass. During the forward pass, the node outputs are computed until Layer 4 and the consequent parameters, which are the coefficients of the over-constrained equation, are obtained using the least squares

estimator (Kalman filtering algorithm) described on page 263. There are a total of 48 consequent parameters and 21 premise parameters for the fuzzy partition described above. These parameters, namely, the width, center, and slope of the generalized bell function, are updated, or tuned, using the gradient descent method. In the backward pass, the error signals propagate backwards and the error at each layer is used to update the premise parameters.

Membership updates The selection of the membership function depends on the application and the quality of data. As mentioned by one of the reviewers of the original ANFIS paper [35], the learning mechanism should not be applied to determine membership functions in the Sugeno ANFIS, since they convey linguistic and subjective descriptions of possibly ill-defined concepts. However, if the size of the data set is large, then fine-tuning of the membership functions is recommended (or even necessary) since human-determined membership functions are seldom optimal in terms of reproducing desired outputs. If the data set is too small, then it probably does not contain enough information about the target system. In such a situation, the membership functions determined might not represent important information pertaining to the data set. In a case like this, it is recommended that the membership functions be fixed throughout the learning process. If the membership functions are fixed and only the consequent part is updated, the Sugeno ANFIS can be viewed as a functional-link network, where the "enhanced representations"of the input variables are obtained via the membership functions. These enhanced representations determined by human experts apparently provide more insight into the target system than the functional expansion or the tensor (outer product) models. By updating the membership functions, we are actually tuning this enhanced representations for better performance [36].

ANFIS results The training data previously formulated to train the backpropagation neural network is used to train the ANFIS and tune the premise and consequent parameters. In the computation of the consequent parameters

$$a_i^j,\ i = 0,\ \cdots,\ 3 \quad \text{and} \quad j = 1,\ \cdots,\ 12$$

using Equation 8.9, the outputs of each layer, until Layer 3, are computed for all patterns. The target vectors **d,** which are the desired value for each trash type, are fuzzy singletons. The desired value during training is 0.2 for bark objects, 0.4 for stick objects, 0.6 for leaf objects, and 0.8 for pepper objects. The consequent parameters are calculated by solving the over-constrained equation to obtain the consequent parameters. Table 8.7 illustrates the final values of the consequent parameters after 200 epochs of training for the 232-partition.

The outputs of Layer 4 are calculated, based on the consequent parameters. The errors backpropagated at each layer are calculated using the gradient descent method. The premise parameters are updated in Layer 1, based on the errors at the input Layer 1. Table 8.8 shows the initial and final values of the premise parameters after 200 epochs of training.

Table 8.7. Final values of consequent parameters: 232-partition

		Consequent Parameters a_i^j			
j	i	0	1	2	3
1		5.6045	−0.0623	5.4067	10.2944
2		0.0746	0.2576	0.2428	−5.9242
3		−1.8363	0.0098	1.3515	−0.7228
4		0.1011	−0.0361	0.4841	2.0547
5		1.2008	−0.0002	−0.3727	−0.4426
6		0.7795	0.0028	0.3548	−1.0790
7		−3.5730	0.0003	−1.9937	4.0406
8		6.4012	0.0003	−28.0326	27.2515
9		5.9735	−0.0002	−4.8844	−3.2546
10		2.1531	−0.0001	1.4765	−4.9520
11		−1.4469	0.0002	2.1095	2.7445
12		−0.2404	0.0001	0.0892	1.3792

Table 8.8. Initial and final values of the premise parameters: 232-partition.

Ling. Var.		Area		Solidity			E^{dif}	
Ling. Val.		Small	Large	Small	Medium	Large	Small	Large
Initial	a	180.0000	4820.0000	0.3333	0.3333	0.3333	0.2500	0.2500
Para-	b	2.0000	40.0000	2.0000	2.0000	2.0000	2.0000	2.0000
meters	c	0.0000	5000.0000	0.0000	0.5000	1.0000	0.0000	0.5000
Final	a	179.9964	4819.9983	0.4730	0.4839	0.0441	0.1097	0.0874
Para-	b	2.2446	40.0018	1.9899	2.0003	2.0442	2.0230	2.0453
meters	c	-0.0010	5000.0017	0.0983	0.5528	0.9362	0.0057	0.3249

Figures 8.15–8.17 show the initial and the final (tuned) membership functions for *area*, *solidity*, and E^{dif} measures. Based on the premise and consequent parameters, the training data is classified to evaluate the classification results.

The decision procedure for the identification of the trash types is as follows: The input pattern for each object in the test sample is passed in the forward direction. With the final premise parameters defining the membership functions, the outputs at each layer are calculated. The output y^* which is the sum of the outputs of Layer 4 is calculated. The outputs of the adaptive nodes at Layer 4 use the updated consequent parameters. Based on the output y^*, the input test pattern is classified to belong to a particular trash type. If the output is in the range of the target value $\pm$ 0.1, then the trash type is classified to belong to that trash type. For example, if the output of the network lies in the range of 0.2 ± 0.1, it is classified as bark, since 0.2 is the desired value of bark objects. For example, the output for object 16 (pattern 9) of test sample #1 (TA114) is 0.2777 and is classified as a bark object (Table 8.10).

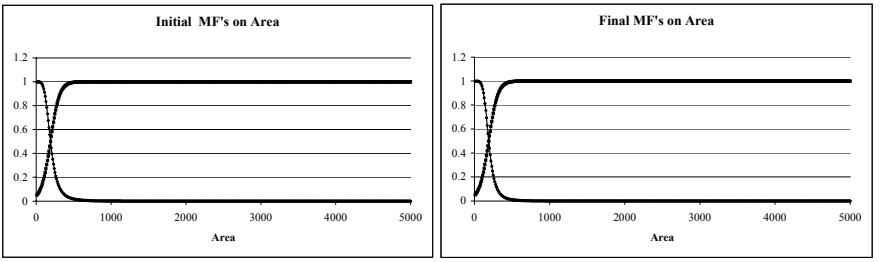


Figure 8.15. Initial and final membership functions on *area*: 232-partition

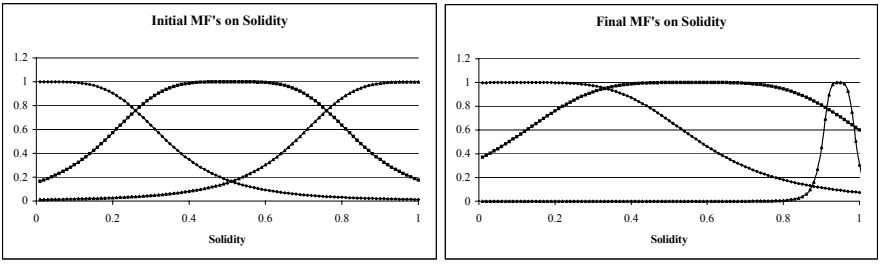


Figure 8.16. Initial and final membership functions on *solidity*: 232-partition

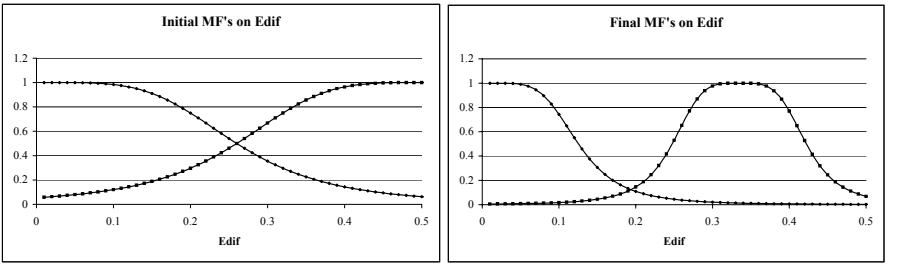


Figure 8.17. Initial and final membership functions on E^{dif}: 232-partition

The classification results of the training data are illustrated in Table 8.9. The classification results indicated superior classification rates with ANFIS compared to the fuzzy clustering and the backpropagation neural network. A total of 5 trash objects out of 213 trash objects were misclassified resulting in a classification rate of 97.653%.

Table 8.9. Classification results of trash objects in the training data using ANFIS: 232-partition

Total number of objects	Actual trash type	Classified trash type			
		Bark	Stick	Leaf	Pepper
18	Bark	16	2	0	0
34	Stick	0	33	1	0
54	Leaf,	0	1	53	0
107	Pepper	0	0	1	106

Classification results of test samples using ANFIS The classification of various trash objects in test sample #1 (TA114) is illustrated in Table 8.10.

Table 8.10. Classification results of trash objects in test sample #1 (TA114) using ANFIS: 232-partition

Pattern	Blob Id	Area	Solidity	E^{dif}	Net output	Actual trash	Classified trash
1	1	22	0.9565	0.1222	0.7952	Pepper	Pepper
2	3	100	0.9346	0.1172	0.8089	Pepper	Pepper
3	4	17	1.0000	0.0283	0.7927	Pepper	Pepper
4	6	28	1.0000	0.0952	0.8116	Pepper	Pepper
5	7	124	0.9688	0.2085	0.7305	Pepper	Pepper
6	9	113	0.9658	0.0395	0.8081	Pepper	Pepper
7	10	13	0.8125	0.0867	0.9166	Pepper	Pepper
8	13	84	0.7568	0.3180	1.7399	Stick	Pepper
9	16	491	0.5696	0.1861	0.2777	Bark	Bark
10	20	11	1.0000	0.2292	0.7963	Pepper	Pepper
11	22	46	0.6479	0.0621	1.3404	Pepper	Pepper
12	23	128	0.9343	0.1389	0.7993	Pepper	Pepper
13	26	11	1.0000	0.0000	0.7815	Pepper	Pepper
14	28	155	0.8908	0.2325	0.7408	Pepper	Pepper
15	31	42	0.9545	0.0562	0.8004	Pepper	Pepper
16	34	16	0.8421	0.0381	0.8223	Pepper	Pepper
17	35	49	0.6622	0.0900	1.3191	Pepper	Pepper
18	36	17	1.0000	0.1133	0.8045	Pepper	Pepper
19	47	31	0.7209	0.0870	1.1874	Pepper	Pepper
20	50	34	0.9444	0.0984	0.7982	Pepper	Pepper
21	51	126	0.8873	0.2447	0.7873	Pepper	Pepper
22	55	111	0.9407	0.2265	0.7727	Pepper	Pepper
23	56	536	0.5714	0.2496	0.4998	Bark	Stick
24	57	70	0.9091	0.0696	0.8050	Pepper	Pepper
25	58	75	0.9494	0.0393	0.8065	Pepper	Pepper
26	60	11	1.0000	0.0764	0.7952	Pepper	Pepper
27	66	19	0.8261	0.2060	0.8072	Pepper	Pepper
28	68	23	0.9200	0.1095	0.7966	Pepper	Pepper
29	71	19	0.9500	0.0452	0.7913	Pepper	Pepper
30	75	22	0.8148	0.3951	0.9282	Pepper	Pepper
31	76	13	0.8667	0.0217	0.7838	Pepper	Pepper

The classification results summarized in Table 8.11 for the five test samples are unsatisfactory for bark, stick, and leaf objects in the samples. Even though the classification rate is 91.8367%, the classification of the individual trash types is poor — especially for bark, stick, and leaf objects. Due to the presence of pepper in high numbers in the cotton samples, and since all three classifiers classify

pepper objects at high accuracies, the computation of the classification rates might be skewed as a classification criterion. However, it should be mentioned that based on the classification results of the training data, it is evident that it is possible to obtain superior classification rates for objects that prescribe the typical shapes that define the trash types. Also as mentioned previously, most of the misclassification is due to segmentation defects. Research efforts with segmentation techniques and acquisition of cotton images need to be addressed in the future.

Table 8.11. Classification results of trash objects in five test samples using ANFIS: 232-partition

Total number of objects	Actual trash type	Classified trash type			
		Bark	Stick	Leaf	Pepper
10	Bark	3	4	3	0
4	Stick	0	1	0	3
16	Leaf	1	2	6	7
215	Pepper	0	0	0	215

In order to evaluate the performance of the ANFIS, different membership partitions were tested. The classification accuracy of the trash types with the use of triangular and trapezoidal membership functions was also investigated. The best classification results were obtained with the membership partition as shown in Figures 8.18–8.20 for the three features.

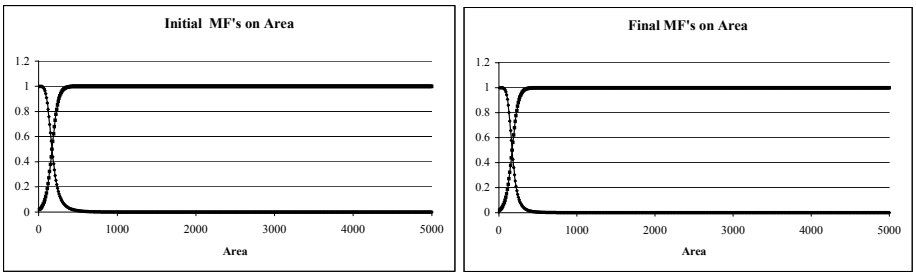


Figure 8.18. Initial and final membership functions on *area*: 222-partition

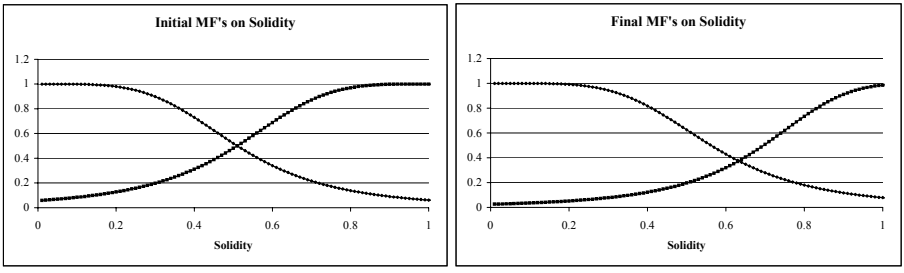


Figure 8.19. Initial and final membership functions on *solidity*: 222-partition

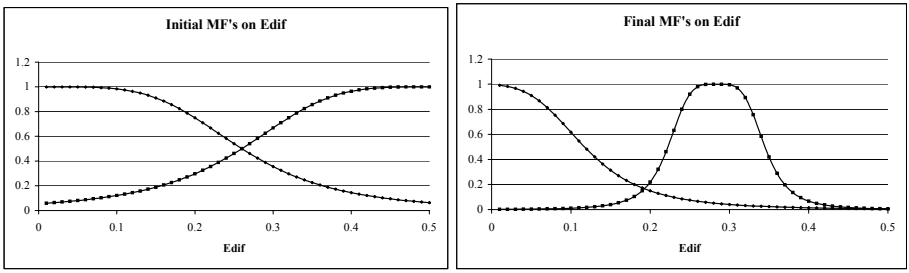


Figure 8.20. Initial and final membership functions on E^{dif}: 222-partition

The linguistic variable *solidity* was partitioned as *small* and *large* instead of *small*, *medium*, and *large* (membership partition-232). There are a total of 8 fuzzy rules for identifying the trash types; they are summarized in Table 8.12. Tables 8.13 and 8.14 are the updated values of the consequent and premise parameters after 200 epochs of training with the 222-membership partition.

Table 8.12. TSK fuzzy rules for trash identification with 222-partition[2]

Rule	If *area* is	and *solidity* is	and E^{dif} is	then *trash type* is
R^1	Small	Small	Small	Pepper
R^2	Small	Small	Large	Pepper
R^3	Small	Large	Small	Pepper
R^4	Small	Large	Large	Pepper
R^5	Large	Small	Small	Bark
R^6	Large	Small	Large	Bark
R^7	Large	Large	Small	Leaf
R^8	Large	Large	Large	Stick

Table 8.13. Final values of consequent parameters: 222-partition

		Consequent Parameters a_i^j			
j	i	0	1	2	3
1		2.2184	−0.0016	−0.6010	3.5199
2		−0.0841	0.0069	1.1953	10.1662
3		−0.0897	0.0010	0.7641	−0.5085
4		−1.2554	0.0004	2.1392	−1.8541
5		0.1818	0.0000	−0.1468	−0.1665
6		3.0904	0.0000	−7.5783	4.1122
7		0.8847	−0.0001	−0.1512	−0.1263
8		1.9480	0.0000	−0.7704	−1.0734

[2]The membership partition of the input variable *area*, *solidity,* and E^{dif} are {(Small, Large), (Small, Large), (Small, Large)} and is represented as 222-partition in the text.

Table 8.15 illustrates the classification results of the training data. As seen from the table, the ANFIS performance is excellent in identifying the trash types. Of the 18 bark objects, 2 objects were misclassified, one as stick and one as leaf. Of the 34 stick objects, 2 were classified as leaf. Only 1 object out of 54 leaf objects was classified as stick. Similarly, only 1 object is classified as leaf out of the 107 pepper objects. This classification rate with the 222-membership partition is 97.1831%.

Table 8.14. Initial and final values of the premise parameters: 222-partition

Linguistic variable		Area		Solidity		E^{dif}	
Linguistic value		Small	Large	Small	Large	Small	Large
Initial parameters	a	160.0000	4840.0000	0.5000	0.5000	0.2500	0.2500
	b	2.0000	60.0000	2.0000	2.0000	2.0000	2.0000
	c	0.0000	5000.0000	0.0000	1.0000	0.0000	0.5000
Final parameters	a	160.0040	4839.9914	0.6568	0.3598	0.1485	0.0622
	b	2.4631	59.9931	1.9984	1.9849	2.0060	2.0493
	c	0.0090	5000.0087	-0.0444	1.0529	-0.0332	0.2732

Table 8.15. Classification results of trash objects in training data using ANFIS: 222-partition

Total number of objects	Actual trash type	Classified trash type			
		Bark	Stick	Leaf	Pepper
18	Bark	16	1	1	0
34	Stick	0	32	2	0
54	Leaf	0	1	53	0
107	Pepper	0	0	1	106

The identification results of the trash objects in 5 test samples with the 222-partition are illustrated in Table 8.16. The classification results are superior compared to all other classifiers discussed so far. The classification rates for leaf and pepper objects are excellent. However, the classification of bark and stick objects is satisfactory. As previously mentioned, one of the reasons for the misclassification of bark and stick objects in the test samples is due to segmentation defects. The stick object in test sample #1 is a single object but segmented as a stick and pepper object. During classification, both of these objects are classified as pepper objects. The stick object is partially buried under the cotton, resulting in discontinuities of the object in the segmented image.

Based on the classification results of the training data, it is seen that the ANFIS gives the best classification results with excellent classification rates. Although the classification accuracy of some of the trash types is low, with proper segmentation it is possible to identify the trash types with increased accuracies. The results from the ANFIS indicate that the two sets of membership partitions, namely, 232-partition and 222-partition, classify the trash types at similar classifications rates. This is illustrated in Table 8.17.

Table 8.16. Classification results of trash objects in five test samples using ANFIS: 222-partition

Total number of objects	Actual trash type	Classified trash type			
		Bark	Stick	Leaf	Pepper
10	Bark	6	4	0	0
4	Stick	0	2	0	2
16	Leaf	0	2	14	0
215	Pepper	0	0	0	215

Table 8.17. Classification rates with 232- and 222-partition

Membership Partition	232-Partition		222-Partition	
Samples	Training	Test	Training	Test
Classification rate (%)	97.6526	91.8367	97.1831	95.1020

From the results of this application, we conclude that ANFIS produces far superior results in terms of its performance as a classifier, and has potential for on-line implementation based upon its ability to adapt. Due to the "fuzzy" nature of the variables, the ANFIS architecture is ideally suited for applications that require rule-based reasoning as part of the decision-making process. The adaptive nature of the network architecture makes ANFIS a highly robust tool for decision-making under uncertainty.

8.5 Integrated pest management systems

Control of pesticides in agricultural ecosystems is essential towards minimizing environmental pollution. Lowering the use of pesticides requires the implementation of biological control wherein natural enemies and predators are used to control the population of insects that are harmful to the agricultural commodity. Therefore, the goal in Integrated Pest Management (IPM) systems is to implement methodologies that can minimize pesticide use while maximizing the use of biological controls. Achieving this goal requires carefully monitoring the populations of specific insect species and determining the appropriate mix of pesticide and biological controls for the target agricultural ecosystem. In this context, there is a need to develop methods to identify the class of insects that populate specific agricultural ecosystems. In this example, we discuss a neuro-fuzzy approach to insect classification.

Statistical approaches to classifying insects have been met with limited success. This is due primarily to the fact that the spatial patterns of insects are not fixed. This problem is made even more complex by changes in insect population with season and time. Factors such as weather, host plants, soil, predator, parasitoid, and behavioral factors of the insect population can all contribute towards an extremely complex problem in insect classification. Hence it is difficult, if not impossible, to obtain satisfactory statistical models to predict or classify insect categories. Figure 8.21 illustrates a framework for developing IPM systems.

Referring to Figure 8.21, the collection of insects is a random sample that is representative of the various insect types that may be present in the agricultural setting such as a large cotton farm. As such, statistical methods for analyzing the data could be applied. Samples of insects can be collected from various parts of a farm, and population estimates of each class of arthropod can be obtained.

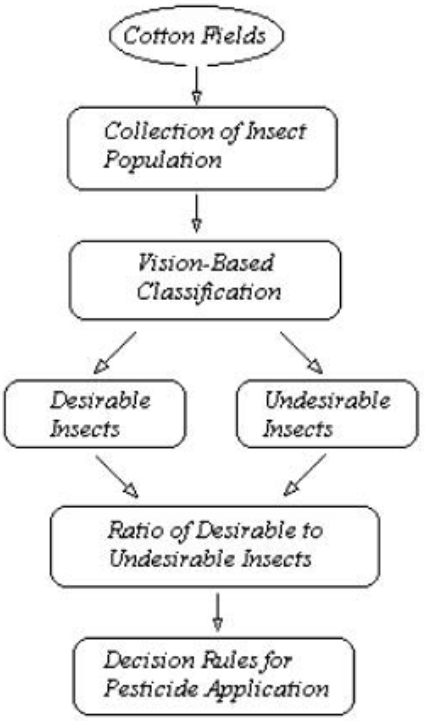


Figure 8.21. Integrated pest management scheme

Such estimates may be easy to obtain for small farm areas, but certainly present a formidable task for large farms that could be thousands of acres in size. For this reason, we need some automated means of collecting insects and analyzing the sampled collection for various classes of insects. For this, a computer vision-based approach can be used for classifying insects into two basic classes, namely, desirable and undesirable insects. By desirable, we mean insects that belong to the predator species, and undesirable implying pests.

In addition, we are also suggesting that each of the basic classes be further divided into specific insect types that form the biological control species and pest species, respectively. Intuitively then, a ratio of desirable to undesirable insects should provide a basis for developing decision rules for pesticide application. It is clear that these decision rules would be fuzzy "If...then..." rules. For example, a typical set of fuzzy rules might be of the form

- If ratio of insects is small and pest A is high, then apply high concentration pesticide Z.

- If ratio of insects is medium and pest A is medium, then apply medium concentration pesticide Z.

While this example does not specifically discuss the formulation of such decision-making rules, we are merely suggesting that it is conceivable that such rules can be generated. The significant attributes of such decision rules is based upon the accuracy of insect classification and how well we can actually determine the pest populations in terms of specific species.

Note also that the objective is to perform such classification in an automated manner with little or no human intervention. The intent is to develop field deployable IPM systems where there is little or no access to experts in entomology. As such, the IPM system is expected to provide the expertise to the farmer directly in an agricultural setting. Classical statistical methods fail to provide the necessary basis for such an approach to developing successful IPM systems.

There are several important functions that are desired in a vision-based classification system. First, an imaging system is needed to obtain high resolution images of the objects requiring classification. This can be obtained using a high resolution digital camera.

The second function is an image processing system that is capable of segmenting the original image, performing thresholding to eliminate debris and other unwanted material in the sample collection, and providing images of insects at a high enough resolution to allow appropriate feature extraction.

The third function is the classification system. Statistical methods suffer from their inability to provide a model that can account for the large variations that can occur in the insect morphology. Nontraditional approaches such as artificial neural networks, fuzzy logic, and possibly other hybrid approaches need to be investigated for classifier design.

Classifier design Pattern classification is generally considered to be a high-end task for computer-based image analysis. The complexities range from locating and recognizing isolated known objects, recognizing objects in poorly defined classes, to much more open-ended problems of recognizing possible overlapping objects or classes. The idea therefore is to select an appropriate set of features that is common to all object classes and that can provide discrimination between various classes. Ideally, one would expect the features to be distinct for each class in order to obtain "perfect" classification. Obviously, in a less than ideal environment such "perfect" classification cannot be obtained.

Table 8.18 provides a list of insects that inhabit cotton and alfalfa agricultural ecosystems, and that are referred to by their common names.

Table 8.18. List of insects in cotton and alfalfa fields

Insect Type	Good	Bad	Comments
Assassin bug	X		Harmful to humans
Big-eyed bug	X		Harmless
Green lacewing adult	X		Harmless
Lacewing larva	X		Harmless
Hippodamia lady beetle adult	X		Harmless
Hippodamia ladybug larva	X		Harmless
Nabid adult	X		Harmless
Stinkbug adult	X		Harmless
Collops beetle	X		Harmless
Leaf hopper		X	Can be destructive
Lygus adult		X	Destructive
Three-corned alfalfa hopper		X	Destructive
Cucumber beetle		X	Destructive

The list includes insects that have a destructive effect on the crop, and others that are harmless to cotton plants but are considered biological predators. A few of the insect types listed in Table 8.18 are shown in Figures 8.22–8.24.

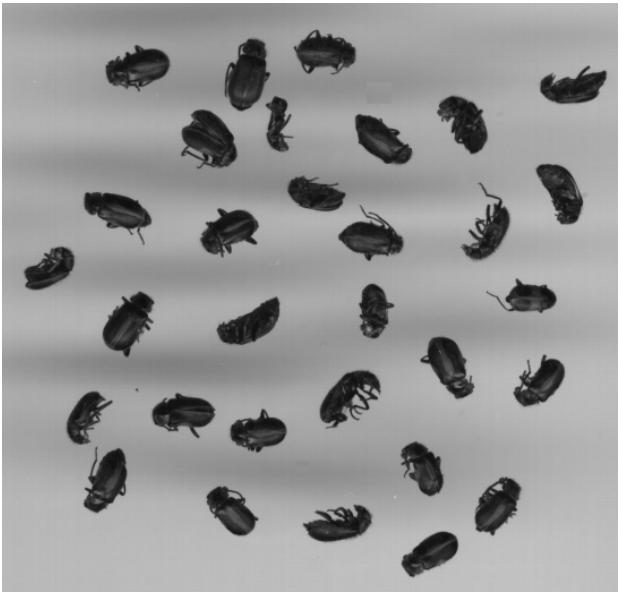


Figure 8.22. Collops beetle

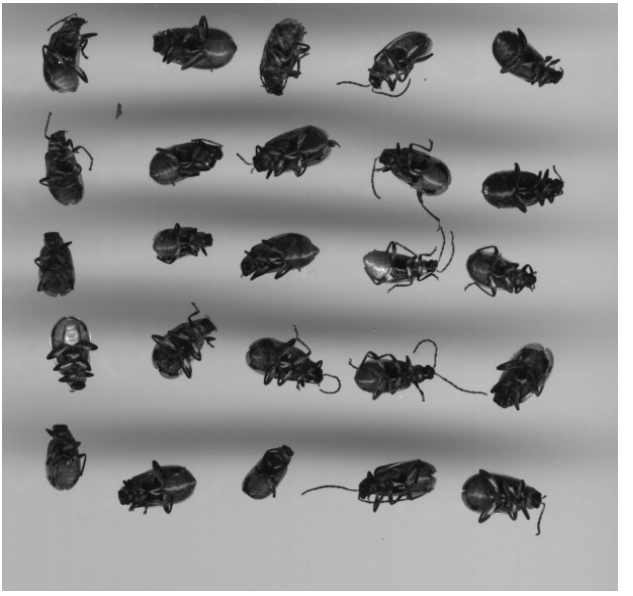


Figure 8.23. Cucumber beetle

Figure 8.24. Lygus adult

The type of features used in a classifier design dictates classifier performance. If the features are distinct, then one can expect excellent performance in terms of classification. For objects of the type in this example, one has to rely on

empirical relationships that describe the features of objects. Size, shape, and other qualitative descriptors provide some level of discrimination between objects. Selection of these descriptors requires some judgment and testing to see how well the classifier can be trained to provide a high level of classification. The descriptors used in this example should by no means be assumed to represent a unique set. There may be other descriptors, a combination of which could produce better results. This however is left as an exercise for those attempting to pursue further studies in this area.

The features used in this example are computed using the following empirical relationships:

1. Form Factor (FF) $= (4\pi A/P^2)$
2. Roundness Factor (RF) $= 4A/\pi D_{\max}^2$
3. Aspect Ratio (AR) $= mD_{\max}/D_{\min}$
4. Compactness Factor (CF) $= \sqrt{(4/\pi)A}/D_{\max}$

where A is the area of an object in pixels, P is the perimeter of an object in pixels, and $D_{\max}$ and $D_{\min}$ are lengths of the major and minor axes of an ellipse fitted around an object. Note that the Aspect Ratio is 1.0 for a circular object.

Table 8.19 provides a set of typical features computed for 13 insect classes.

Table 8.19. Typical feature set for various insect classes

	Features				
Insect Type	AR	P	FF	RF	CF
Assassin bug	2.4773	109	2.5156	0.2549	0.5049
Big-eyed bug	2.0556	37	4.9074	0.4972	0.7051
Collops beetle	1.7586	51	5.2493	0.5319	0.7293
Cucumber beetle	1.7000	85	5.0970	0.5164	0.7186
Green lacewing adult	4.5000	99	1.8056	0.1829	0.4277
Hippodamia lady beetle adult	1.4375	69	6.4607	0.6546	0.8091
Hippodamia ladybug larva	2.1935	68	3.6090	0.3657	0.6047
Leaf hopper	3.1071	87	2.7873	0.2824	0.5314
Lacewing larva	2.5000	60	3.4566	0.3502	0.5918
Lygus adult	2.4400	61	3.6473	0.3696	0.6079
Nabid adult	2.8000	70	3.1217	0.3163	0.5624
Stinkbug adult	1.4028	101	6.7639	0.6853	0.8278
Three-corned alfalfa hopper	1.6136	71	5.0224	0.5089	0.7134

Clustering approach to classifier design Figure 8.25 conceptually illustrates clusters of class objects in two-dimensional feature space.

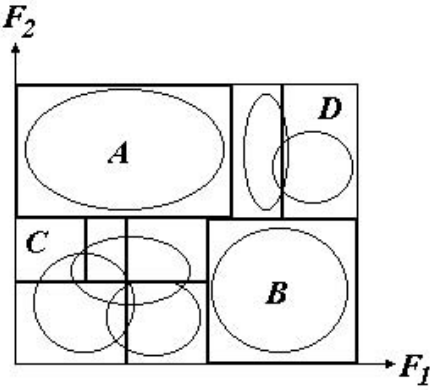


Figure 8.25. Clustering approach

Referring to Figure 8.25, while class objects A and B are clearly separable, clusters C and D represent groups of class objects that overlap and are therefore inseparable. Features representing class types A and B, that are distinct, can easily be trained using the classical feedforward multi-layered neural networks whereas, objects in clusters C and D can only be determined using fuzzy logic-based approaches. In a fuzzy approach, the clusters are partitioned optimally, and a set of fuzzy "If... then..." rules are generated. These rules provide a basis for pattern classification using ANFIS.

Fuzzy subsets, in the form of membership functions used in ANFIS, are shown in Figure 8.26.

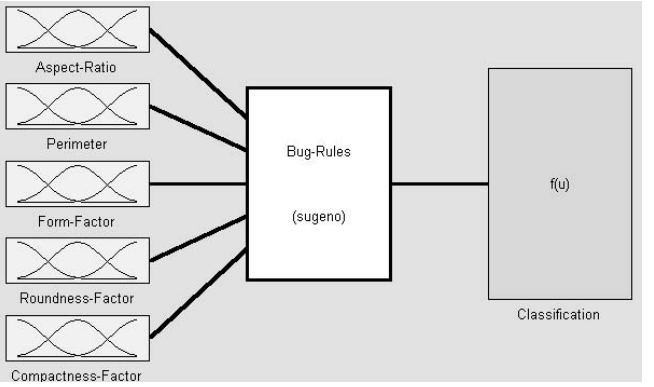


Figure 8.26. Classification scheme

The features listed in Table 8.19 are typical of the input variables needed to train ANFIS. A total of 222 training patterns from 13 different insect species were used to train ANFIS. Figures 8.27–8.36 show the membership functions before and after training. The range over which each of the input variables is defined was obtained by looking at the minimum and maximum values of each

feature. Because only two linguistic variables are used for each feature, a total of 32 rules is generated by the ANFIS fuzzy inference system.

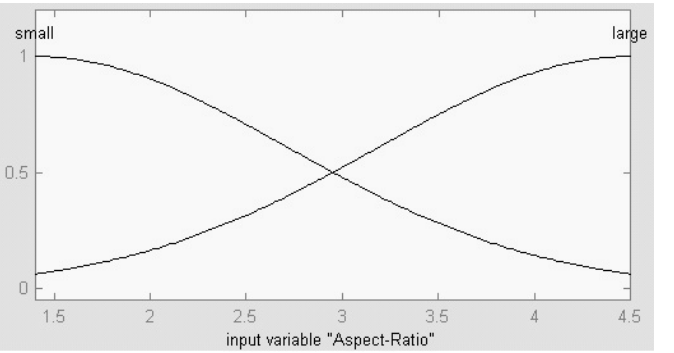


Figure 8.27. Aspect-ratio before training

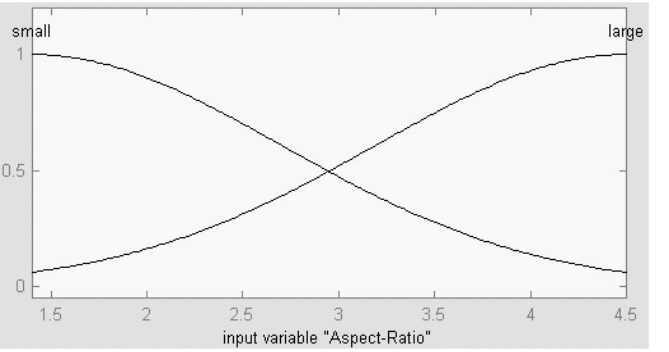


Figure 8.28. Aspect-ratio after training

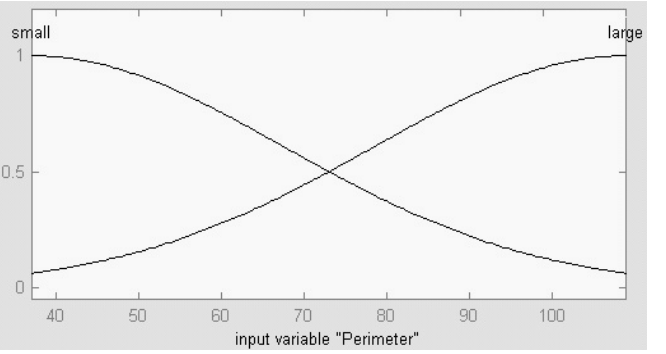


Figure 8.29. Perimeter before training

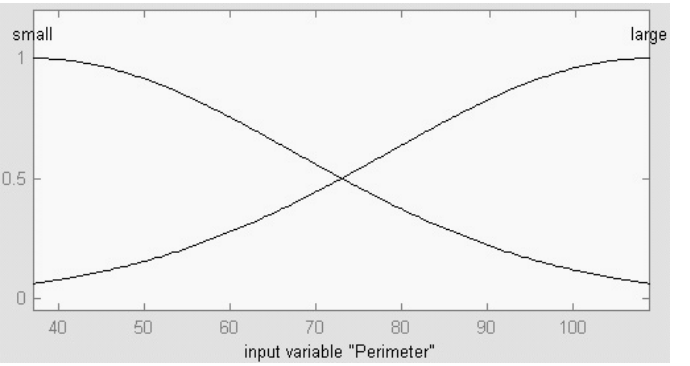


Figure 8.30. Perimeter after training

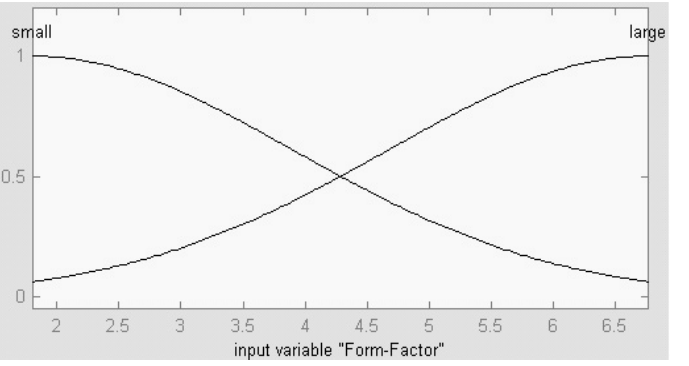


Figure 8.31. Form-factor before training

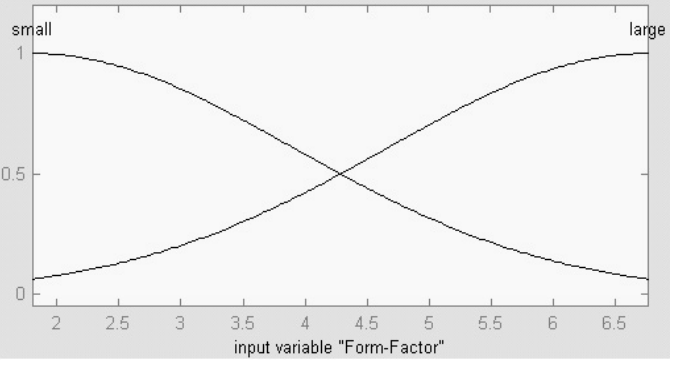


Figure 8.32. Form-factor after training

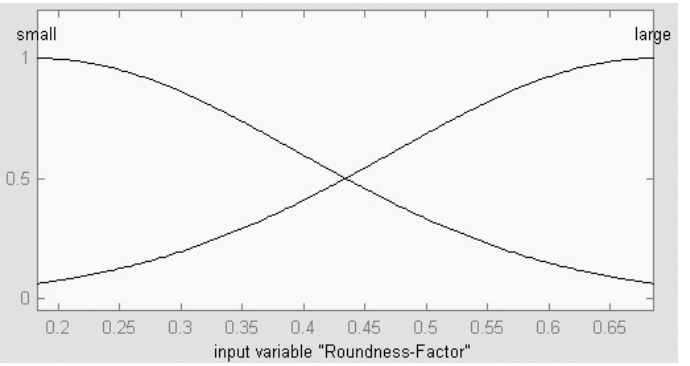


Figure 8.33. Roundness-factor before training

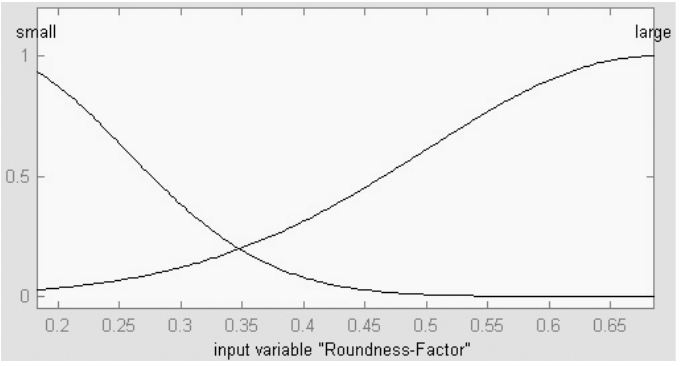


Figure 8.34. Roundness-factor after training

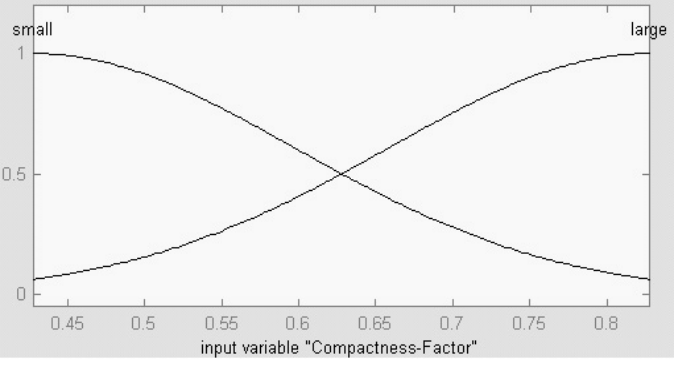


Figure 8.35. Compactness-factor before training

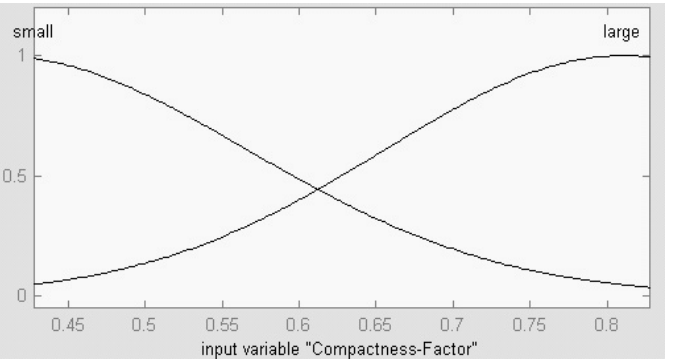


Figure 8.36. Compactness-factor after training

Figure 8.37 illustrates the performance of ANFIS to a set of test data comprising 222 samples for the 13 classes of insects listed in Table 8.19. From this analysis, it appears that ANFIS provides robust performance as a classifier. Further details of this application can be found in [25].

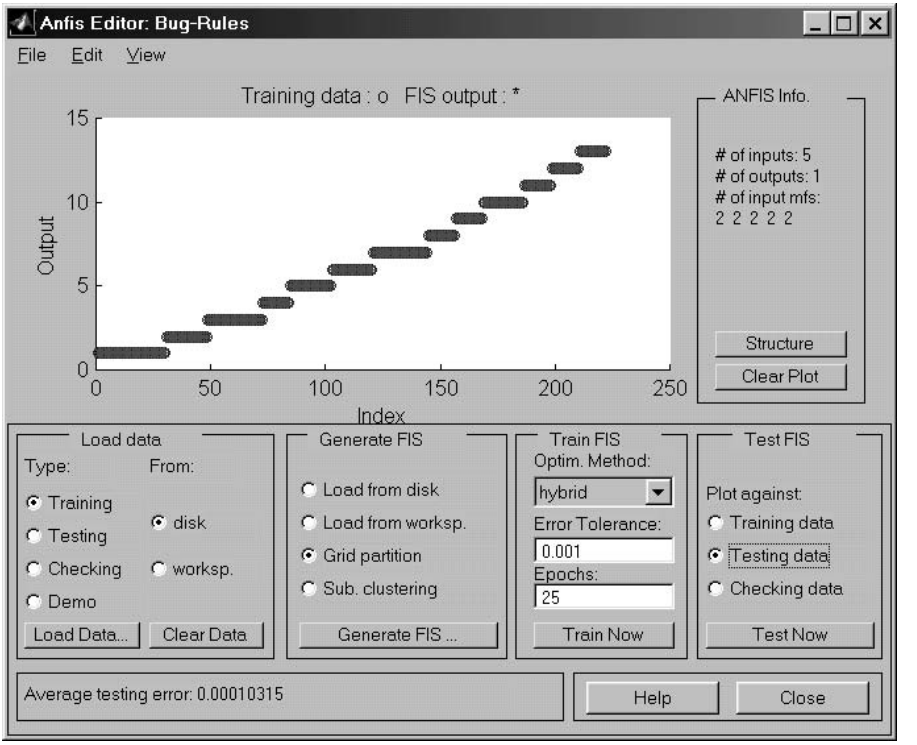


Figure 8.37. ANFIS performance to test data

8.6 Comments

The applications discussed in this chapter clearly illustrate the diversity of topics that allow soft computing as a sure means for solving an otherwise difficult problem. Color matching, trash identification in cotton, insect identification in agricultural ecosystems, and control of thermoelectric cooling of laser materials are all examples of highly nonlinear problems for which no other technology has shown the potential of providing favorable solutions. Our goal in this final chapter has been to provide detailed discussions of how problems with no standard mathematical model can be addressed with human-like reasoning. The examples and discussions are intended to motivate the reader to pursue solutions of far more complex problems in the fields of medical diagnosis; socio-economic forecasting; weather modeling and forecasting; biological and chemical identification and detection; and a multitude of other fields where human knowledge can be transformed into "models" for machine learning. Adaptation and robustness ultimately dictate how well machines can perform human-like tasks that may lead us toward the successful development and deployment of systems with self-diagnostic and self-repair capabilities.

Bibliography

[1] S. Abe and M.S. Lan, A method for fuzzy rule extraction directly from numerical data and its application to pattern recognition, *IEEE Trans. on Fuzzy Systems*, 3, 18-28, 1995.

[2] H. Adeli and S.L. Hung, *Machine Learning: Neural Networks, Genetic Algorithms and Fuzzy Systems*, John Wiley & Sons, New York, 1995.

[3] S. Amari and N. Kasabov, eds., *Brain-Like Computing and Intelligent Information Systems*, Springer-Verlag, Berlin, 1998.

[4] M. Anthony and P.O. Bartlett, *Neural Network Learning: Theoretical Foundations*, Cambridge University Press, Cambridge, 1999.

[5] S. Barnett and R.G. Cameron, *Introduction to Mathematical Control Theory*, 2nd Edition, Clarendon Press, Oxford, 1985.

[6] J.C. Bezdek and S.K. Pal, eds., *Fuzzy Models for Pattern Recognition*, IEEE Press, Piscataway, NJ, 1992.

[7] F.W. Billmeyer, Jr. and M. Saltzman, *Principles of Color Technology,* John Wiley & Sons, New York, 1966.

[8] C.M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[9] G. Bojadziev and M. Bojadziev, *Fuzzy Sets, Fuzzy Logic, Applications*, World Scientific, London, 1995.

[10] B. Bouchon-Meunier, V. Kreinovich, H.T. Nguyen, "Non-Associative Operations," *Proceedings of the Second International Conference on Intelligent Technologies (InTech 2001)*, Faculty of Science and Technology, Assumption University, Bangkok, Thailand, 39-46, 2001.

[11] M. Brown and C. Harris, *Neurofuzzy Adaptive Modelling and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[12] P.K. Cheo, ed., *Handbook of Solid-State Lasers*, Marcel Dekker, Inc., New York, 1989.

[13] A.K. Cousins, Temperature and thermal stress scaling in finite-length end-pumped laser rods, *IEEE J. Quantum Electron.*, 28(4), 1057-1069, 1992.

[14] E. Czogała and J. Łęski, *Fuzzy and Neuro-Fuzzy Intelligent Systems*, Springer-Verlag, Berlin, 2000.

[15] J. Dayhoff, *Neural Network Architectures: An Introduction*, Van Nostrand Reinhold Co., New York, 1990.

[16] J.J. D'Azzo and C.H. Houpis, *Linear Control Systems Analysis and Design – Conventional and Modern*, McGraw-Hill, New York, 1995.

[17] C.W. de Silva, *Intelligent Control: Fuzzy Logic Applications*, CRC Press, Boca Raton, FL, 1995.

[18] R.C. Dorf, ed., *The Electrical Engineering Handbook*, CRC Press, Boca Raton, FL, 1993.

[19] K. Dutton, S. Thompson, and B. Barraclough, *The Art of Control Engineering*, Addison-Wesley Longman, Ltd., Essex, UK, 1997.

[20] R. Eberhart and B. Dobbins, *Neural Network PC Tools: A Practical Guide*, Academic Press, New York, 1990.

[21] J. Farrell, Neural control systems, invited chapter in *The Controls Handbook*, W. S. Levine, ed., CRC Press, Boca Raton, FL, 1017-1030, 1996.

[22] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*, Prentice-Hall, Upper Saddle River, NJ, 1994.

[23] T.L. Fine, *Feed Forward Neural Network Methodology*, Springer-Verlag, Berlin, 1999.

[24] R. Fuller, *Introduction to Neuro-Fuzzy Systems*, Advances in Soft Computing Series, Springer-Verlag, Berlin, 2000.

[25] H. Gassoumi, *A Soft Computing Approach for Classification of Insects in Agricultural Ecosystems*, Ph.D. dissertation, New Mexico State University, Las Cruces, NM, 2000.

[26] M. Gehrke, C.L. Walker, and E.A. Walker, Normal forms and truth tables for fuzzy logics, chapter in *Proceedings of Linz2000 Conference*, *Fuzzy Sets and Systems*, to appear.

[27] I.R. Goodman, H.T. Nguyen, and E.A. Walker, *Conditional Inference and Logic for Intelligent Systems: A Theory of Measure-Free Conditioning*, North-Holland, Amsterdam, 1991.

[28] G.C. Goodwin, S.F. Graebe, and M.E. Salgado, *Control System Design*, Prentice-Hall, Upper Saddle River, NJ, 2001.

[29] S. Grossberg, *Neural Networks and Natural Intelligence*, MIT Press, Boston, 1988.

[30] F.M. Ham and I. Kostanic, *Principles of Neurocomputing for Science and Engineering*, McGraw-Hill, New York, 2001.

[31] M.H. Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press, Boston, 1995.

[32] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Co. Inc., New York, 1994.

[33] F. Höppner, F. Klawonn, and R. Kruse, *Fuzzy Cluster Analysis*, John Wiley & Sons, New York, 1999.

[34] M.E. Innocenzi et al., Thermal modeling of continuous-wave end-pumped solid-state lasers, *Appl. Phys. Lett.*, 56(19)(5), 1831-1833, May 1990.

[35] J.S.R. Jang, ANFIS: adaptive-network-based fuzzy inference system, *IEEE Transactions on Systems, Man and Cybernetics*, 23(3), 665-684, 1993.

[36] J.S.R. Jang, C. T. Sun, and E. Mituzani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[37] S.V. Kartalpoulos, *Understanding Neural Networks and Fuzzy Logic*, IEEE Press, Piscataway, NJ, 1996.

[38] W. Koechner, *Solid-State Laser Engineering*, 3rd Edition, Springer-Verlag, Berlin, 1992.

[39] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, Upper Saddle River, NJ, 1992.

[40] C.-T. Lin and C.S. George Lee, *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*, Prentice-Hall, Upper Saddle River, NJ, 1996.

[41] R.J. Marks II, ed., *Fuzzy Logic Technology, and Applications*, IEEE Technology Update Series, IEEE Press, Piscataway, NJ, 1994.

[42] J.L. McClelland and D.E. Rumelhart, *Explorations in Parallel Distributed Processing*, MIT Press, Cambridge, MA, 1988.

[43] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. of Math. Biophysics*, 5, 115-133, 1943.

[44] *Melcor Thermoelectric Handbook*, http://www.melcor.com/handbook.htm, Melcor Corp., Trenton, NJ, 1995.

[45] T.M. Mitchell, *Machine Learning*, McGraw-Hill, New York, 1997.

[46] K.S. Narendra, Intelligent Control, *IEEE Control Systems*, 39-40, January 1991.

[47] K.S. Narendra and S. Mukhopadhyay, Intelligent Control Using Neural Networks, in *Intelligent Control Systems: Theory and Applications*, M.M. Gupta and N.K. Sinha, eds., IEEE Press, Piscataway, NJ, 151-186, 1996.

[48] B. Neuenschwander, et al., Determination of the thermal lens in solid-state lasers with stable cavities, *IEEE J. Quantum Electron.*, 31(6), 1082-1087, 1995.

[49] M.B. Nevel-Son, *Stochastic Approximation and Recursive Estimation*, American Mathematical Society, Providence, RI, 1976.

[50] H.T. Nguyen, M. Sugeno, and R. Yager, eds., *Theoretical Aspects of Fuzzy Control*, John Wiley & Sons, New York, 1995.

[51] H.T. Nguyen, N.R. Prasad, V. Kreinovich, and H. Gassoumi, Intelligent mining in image databases, with applications to satellite imaging and to web search, in *Data Mining and Computational Intelligence*, Kandel et al., eds., Springer-Verlag, Berlin, 2000.

[52] H.T. Nguyen and E.A. Walker, *A First Course in Fuzzy Logic*, 2nd Edition, CRC Press, Boca Raton, FL, 1999.

[53] N.S. Nise, *Control Systems Engineering*, 3rd Edition, John Wiley & Sons, New York, 2000.

[54] M. NØrgaard, O. Ravn, N.K. Poulsen, and L.K. Hansen, *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook*, Springer-Verlag, Berlin, 2000.

[55] K.M. Passino and S. Yurkovich, *Fuzzy Control*, Addison-Wesley Longman, Menlo Park, CA, 1998.

[56] W. Pedrycz, *Fuzzy Sets Engineering*, CRC Press, Boca Raton, FL, 1995.

[57] C.L. Phillips and R.D. Harbor, *Feedback Control Systems*, 4th Edition, Prentice-Hall, Upper Saddle River, NJ, 2000.

[58] N.S. Prasad and N.R. Prasad, Fuzzy logic based cooling scheme for laser materials, *SPIE*, 2755, 357-362, 1996.

[59] N.S. Prasad and N.R. Prasad, A fuzzy logic based approach to color quality processing, *SPIE*, 2761, 111-118, 1996.

[60] T.J. Ross, *Fuzzy Logic with Engineering Applications*, McGraw-Hill, New York, 1995.

[61] W. Rudin, *Principles of Mathematical Analysis*, McGraw-Hill, New York, 1976.

[62] R.J. Schalkoff, *Artificial Neural Networks*, McGraw-Hill, New York, 1997.

[63] M. Siddaiah, M.A. Lieberman, S.E. Hughs, and N.R. Prasad, Computation of trash content in ginned cotton using soft computing techniques, *Proceedings of the 42nd Midwest Symposium on Circuits and Systems*, 1, 547-550, 1999.

[64] M. Siddaiah, M.A. Lieberman, S.E. Hughs, and N.R. Prasad, A soft computing approach to classification of trash in ginned cotton, *Proceedings of the Eighth International Fuzzy Systems Association World Congress*, 1, 151-155, 1999.

[65] M. Siddaiah, M.A. Lieberman, S.E. Hughs, and N.R. Prasad, Identification of trash types in ginned cotton using neuro-fuzzy techniques, *1999 IEEE International Fuzzy Systems Conference Proceedings*, FUZZ-IEEE '99, II, 738-743, 1999.

[66] M. Siddaiah, M.A. Lieberman, S.E. Hughs, and N.R. Prasad, Identification of trash types and correlation between AMS and SWCGRL trash content in ginned cotton, *Proceedings of the 2000 Beltwide Cotton Conferences*, 2, 1549-1555, 2000.

[67] M. Siddaiah, *Identification of trash in cotton using soft computing techniques*, Ph.D. Dissertation, New Mexico State University, Las Cruces, NM, 2000.

[68] P.K. Simpson, Fuzzy min-max neural networks: No. 1, Classification, *IEEE Transactions on Fuzzy Systems*, 3, 776-786, 1993.

[69] P.K. Simpson, ed., *Neural Networks Theory, Technology, and Applications*, IEEE Technology Update Series, IEEE Press, Piscataway, NJ, 1996.

[70] E.D. Sontag, *Mathematical Control Theory*, Springer-Verlag, Berlin, 1998.

[71] M. Sugeno, Fuzzy Measure and Fuzzy Integral, *Trans. SICE*, 8(2), 95-102, 1972.

[72] T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Systems, Man and Cybernetics*, 15(1), 116-132, 1985.

[73] K. Tanaka and M. Sugeno, Stability analysis and design of fuzzy control systems, *Fuzzy Sets and Systems*, 45, 135-150, 1992.

[74] K. Uehara and M. Fujise, Fuzzy inference based on families of alpha-level sets, *IEEE-Transactions on Fuzzy Systems,* 1(2), 111-24, May 1993.

[75] M. Umano and Y. Ezawa, Execution of approximate reasoning by neural network, *Proceedings of FAN Symposium,* 267-273, 1991 (in Japanese).

[76] V.N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, Berlin, 2000.

[77] L.X. Wang and J.M. Mendel, Back-propagation fuzzy systems as nonlinear dynamic systems identifier, *Proceedings of the First IEEE International Conference on Fuzzy Systems*, San Diego, CA, March 1992.

[78] L.X. Wang, Fuzzy systems are universal approximators, *Proceedings of the First IEEE Conference on Fuzzy Systems*, San Diego, CA, 1163-1170, March 1992.

[79] P.J. Werbos, An overview of neural networks for control, *IEEE Control Systems*, 40-41, January 1991.

[80] B. Werners, An interactive fuzzy programming system, *Fuzzy Sets and Systems*, 23, 131-147, 1987.

[81] B. Widrow and M.E. Hoff, Jr., Adaptive switching circuits, *IRE WESCON Convention Record*, part 4, 96-104, 1960.

[82] G. Wyszecki and W.S. Stiles, *Color Science, Concepts and Methods, Quantitative Data and Formulas,* John Wiley & Sons, New York, 1967.

[83] R.R. Yager, On a general class of fuzzy connectives, *Fuzzy Sets and Systems*, 4, 235-242, 1980.

[84] T. Yamakawa, Fuzzy controller hardware system, *Proceedings of Second IFSA Conference*, Tokyo, Japan, 1987.

[85] J. Yen and R. Langari, *Fuzzy Logic: Intelligence, Control, and Information*, Prentice-Hall, Upper Saddle River, NJ, 1999.

[86] H.J. Zimmermann, *Fuzzy Sets, Decision Making, and Expert Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1987.

[87] H.J. Zimmermann, *Fuzzy Set Theory and its Applications*, 2nd Edition, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

[88] H.J. Zimmermann and P. Zysno, Latent connectives in human decision making, *Fuzzy Sets and Systems*, 4, 37-51, 1980.

[89] J.M. Zurada, R.J. Marks II, and C.J. Robinson, eds., *Computational Intelligence Imitating Life*, IEEE Press, Piscataway, NJ, 111-118, 1994.