13

EXPERT SYSTEMS AND THE BASICS OF FUZZY LOGIC

This lesson describes an area from Artificial Intelligence: expert systems. We describe the essentials of fuzzy logic in modeling of expert knowledge. We also touch upon the field of fuzzy control; the general methodology of fuzzy control will be given in the next Lesson 14.

13.1. Expert knowledge and how to represent it in the computer: formulation of the problem

In some extrapolation situations, we have an additional expert knowledge. Many practical problems are naturally formulated as optimization problems. If we have a *complete* information about the object, then optimization becomes a well-defined mathematical problem. However, in many practical situations, we only have a *partial* information about the object. In this case, before we do any optimization, we must first *extrapolate* this partial information. In the previous lessons, we considered the case when this partial information comes from measurements or from experiments, i.e., when it takes the form of *patterns* $(x_1^{(p)}, \ldots, x_n^{(p)}, y^{(p)})$, $1 \leq p \leq P$, from which we can extrapolate the function $y = f(x_1, \ldots, x_n)$ that is consistent with these patterns.

In many practical problems, in addition to the patterns, we also have *experts* whose knowledge about the object we can use. In this lesson, we will describe how we can extract and use this expert knowledge, and how continuous mathematics can help.

Why do we need to extract expert knowledge? By definition, an expert is a person who is well knowledgeable about the object. An expert helicopter pilot is a person who can control a helicopter well; an expert doctor is a person who can cure different diseases, etc. If we already have such an expert, what else do we need? The problem is that in every area,

- there are only a *few* top experts, and
- there are *many* problems to be solved.

It is, usually, simply physically impossible to have a top expert for each problem: we cannot have a top surgeon for every surgery, we cannot have a top helicopter pilot for every helicopter in the world. It is therefore desirable to develop a *computer program* that would somehow incorporate the expert knowledge and give advise comparable in quality with the advise of the top experts. In other words, we want to design a computer program that somehow *simulates* the expert. Such programs are called *expert systems*.

Why cannot we extract the function $f(x_1, \ldots, x_n)$ from an expert? At first glance, the problem seems relatively simple: Since the person is a real expert, we simply ask her multiple questions like "suppose that x_1 is equal to 1.2, x_2 is equal to 1.3, ..., what is y?" After asking all these questions, we will get many patterns, from which we will be able to extrapolate the function $f(x_1, \ldots, x_n)$ using one of the known methods. For example, we may ask the medical expert about all possible combinations of symptoms and test results (blood pressure, temperature, etc.), and write down what exactly treatment the expert recommends. However, there are two problems with this idea:

- First, there is a *computational* problem: Since we need to ask a question for each combination of symptoms and test results, we may end up having to ask *too many* questions. This fact makes our idea *difficult* to implement but *not* necessarily *impossible*: Indeed, we only need to do this lengthy questioning once, so we may afford to spend years, if necessary (actually, the design of the first expert systems did take years).
- However, there is another, more serious problem with this idea that makes it, in most problems, *impossible* to implement. Indeed, this idea may sound reasonable until we try applying it to a skill in which practically all adults consider themselves experts: driving a car. If you ask a driver a question like: "you are driving at 55 miles per hour when the car in 30 feet in front

of you slows down to 47 miles per hour, for how many seconds do you hit the brakes?", nobody will give a precise number.

What can we do? We might install measuring devices into a car (or into a driving simulator), and simulate this situation, but the resulting braking times may differ from simulation to simulation.

The problem is not that the expert has some precise number (like 1.453 sec) in mind that he cannot express in words; the problem is that one time it will be 1.39, another time it may be 1.51, etc.

An expert usually expresses his knowledge by using words from natural language. An expert *cannot*, usually, express his knowledge in *precise* numerical terms (such as "hit the brakes for 1.43 sec"), but he *can* formulate his knowledge by using *words* from natural language. For example, an expert can say "hit the brakes for a while".

So, the knowledge that we can extract from an expert consists of statements like "if the velocity is a little bit smaller than maximum, hit the breaks for a while".

There may be also another uncertainty involved, e.g., an expert may say something like "if a patient has a temperature of 100 degrees, and a headache, and ... ⟨several other symptoms⟩, then, most probably, it is a flu". Here, the words "most probably" are the words from natural language that an expert uses to describe his knowledge.

Since our goal is to describe the expert's knowledge inside a computer, we must represent rules formulated in a natural language inside the computer.

Toy example: a thermostat. We will illustrate different methods for representing expert knowledge on another situation in which everyone feels himself an expert: controlling a *thermostat*.

For simplicity, we will consider a *toy*, simplified thermostat in which a thermometer shows the current temperature and a dial allows us to control the temperature:

- turning the dial to the *left* makes it *cooler*;
- turning it to the *right* makes it *warmer*.

The angle on which we turn the dial will be denoted by u.

We will also assume, for simplicity, that we know the comfort temperature T_0 that we try to achieve. Strictly speaking, the input to our control is the temperature T. However, in reality, our control decisions depend not so much on the *absolute* value of T but rather on the *difference* $T - T_0$ between the actual temperature T and the ideal temperature T_0. Since this difference is important, we will use a special notation for it: $x = T - T_0$. Our goal is to describe the appropriate control u for each possible value x, i.e., to describe a function $u = f(x)$.

For such an easy system, we do not need any expert to formulate reasonable rules; we can immediately describe several reasonable control rules:

- If the temperature T is *close* to T_0, i.e., if the difference $x = T - T_0$ between the actual temperature is *negligible*, then no control is needed, i.e., u should also be negligible.
- If the room is slightly overheated, i.e., if x is positive and small, we must cool it a little bit (i.e., u must be negative and small).
- If the temperature is a little lower than we would like it to be, then we need to heat the room a little bit. In other terms, if x is small negative, then u must be small positive.

We can formulate many similar natural rules. For simplicity, in our toy example, we will restrict ourselves to these three. As a result, we get the following three rules:

- if x is negligible, then u must be negligible;
- if x is small positive, then u must be small negative;
- if x is small negative, then u must be small positive.

Toy example re-formulated. Our goal is to make these and similar rules accessible for the computer. Before describing how to do it, let us first reformulate these rules in a way that will somewhat clarify these rules. Namely, in the formulation of these rules, we want to clearly separate the *properties* (like "x is negligible") and *logical connectives* (like "if ... then"). To achieve this separation, let us introduce a shorthand notation for all the properties, and let us use standard mathematical notations for logical connectives:

- $N(x)$ will indicate that x is negligible;
- $SP(x)$ will indicate that x is small positive;
- $SN(x)$ will indicate that x is small negative;
- $A \to B$ is a standard mathematical notation for "if A then B".

In these notations, the above three rules take the form:

$$N(x) \to N(u); \tag{13.1}$$

$$SP(x) \to SN(u); \tag{13.2}$$

$$SN(x) \to SP(u). \tag{13.3}$$

General case. In general, the expert's knowledge about the dependence of y on $x_1, \ldots, x_n$ can be expressed by several rules of the type

If x_1 is A_{r1}, ..., and x_n is A_{rn}, then y is B_r.

Here, $r = 1, \ldots, R$ is the rule number, and A_{ri} and B_r are words from natural language that are used in r-th rule, like "small", "medium", "large", "approximately 1", etc.

- In our toy example, we only had *one* input variable: the temperature T (or, to be precise, the difference x between T and the desired temperature T_0).
- In the general case, we have *several* input variables, so, in addition to the logical connective "if ... then", we need another logical connective "and".

In mathematics, "and" is usually denoted by & (or by $\wedge$). If we use the standard mathematical notation for "if ... then" and "and", we can re-formulate the above rules as follows:

$$A_{r1}(x_1) \& \ldots \& A_{rn}(x_n) \to B_r(y), \tag{13.4}$$

where $r = 1, \ldots, R$. The set of rules is usually called a *rule base*.

Comment. In this text, we will only consider the case when a rules base consists of *straightforward* if–then rules. In reality, experts may have some additional information about the object, which may be formulated in a more complicated form. For example, an expert may formulate several different rules bases, and then formulate "meta-rules" that decide which of the rule bases are applicable.

The problem. The problem is to represent the expert knowledge in a computer and to be able to process this knowledge in a meaningful way.

13.2. Expert system methodology: an outline

Our goal is to represent rule bases. A rule base has a clear structure:

- A rule base consists of rules.
- Each rule, in its turn, is obtained from properties (expressed by words from natural language) by using logical connectives.

In view of this structure, it is reasonable to represent the rule base by first representing the basic elements of the rule base, and then by extending this representation to the rule base as a whole. In other words, it makes sense to follow the following methodology:

- First, we represent the basic properties $A_{ri}(x_i)$ and $B_r(y)$.
- Second, we represent the logical connectives.
- Third, we use the representations of the basic properties and of the logical connectives to get the representations of all the rules.
- Fourth, we combine the representations of different rules into a representation of a rule base.

As a result of these four steps, we get an advising (expert) system. For example, if we apply these four steps to the medical knowledge, we ideally, get a system that, given the patient's symptoms, provides the diagnostic and medical advise. For example, it can say that most probably, the patient has a flu, but it is also possible that he has bronchitis. Such an advice, coming from an expert system, is usually used by a specialist to make a decision.

However, there are situations like helicopter control where there is no time for a human operator to make a decision: we want a system to automatically make a decision based on its own conclusions. For such control situations, we need an *additional*, fifth follow-up step:

- The computer-based system makes a decision.

The resulting five-step methodology is indeed very successful in control. The resulting control (called *fuzzy control*; see Lesson 14) is used in various areas ranging from appliances (camcorders, washing machines, etc.) to automatically controlled subway trains in Japan to cement kilns to regulating temperature within the Space Shuttle.

There are many books that describe fuzzy control. The reader who is interested in learning more about fuzzy methods in general and fuzzy control in particular is advised to start with one of the following textbooks:

- G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications* (Prentice Hall, Upper Saddle River, NJ, 1995); this book gives an *engineering*-oriented introduction;
- H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic* (CRC Press, Boca Raton, Florida, 1997); this textbook is more oriented towards *mathematical* methods.

In the following, we will describe the *basics* all these five steps.

13.3. Representing natural-language properties: main idea

Before we represent a property $P(x)$, let us represent the expert's degree of belief in this property for each x. To represent a property like "x is positive small" ($SP(x)$), we must be able, for every possible value of the temperature difference x, to represent the expert's opinion of whether this particular value x is indeed small. To represent this opinion, we must solve the following problem:

- All the properties that we have encountered in the previous lessons, such as "x is positive", were *crisp*, in the sense that every real number is either positive, or not.
- On the other hand, properties like "x is small" (in which we are interested) are *not crisp*. To be more precise:
  - Some values x are so small that practically everyone would agree that they are small.
  - Some values x are so huge that practically everyone will agree that they are not small.
  - However, for many intermediate values x, it is difficult to decide whether x is small or not, and experts may disagree:
    * For a researcher who performs temperature-sensitive experiments in the lab, the difference of $x = 0.5$ degrees may not be small at all.
    * However, for a living room, even the difference of ± 5 degrees is usually not only small, but even negligible.

  Such non-crisp properties are called *fuzzy*. This term was introduced by Lotfi Zadeh in 1965.

How can we represent fuzzy properties?

- If $P(x)$ is a *crisp* property (like "positive"), then for every x, $P(x)$ is either true or false.

  Representing the corresponding truth value in the computer is easy: "true" is usually represented by 1, and "false" by 0.
- If $P(x)$ is a *fuzzy* property (like "small"), then in general, for a real number x, an expert may not be sure whether x satisfies this property P or not. For example, for "small", the larger the value x, the less confident the expert that x is small. So, for different values x, an expert may have different "degrees of confidence" that x satisfies this property P:
  - For some values x, the expert may be absolutely sure that the statement $P(x)$ is true. In this case, the degree of confidence corresponds to "true".
  - For some other values x, the expert may be absolutely sure that the statement $P(x)$ is false. In this case, the degree of confidence corresponds to "false".

– For yet other values x, the expert is neither sure that $P(x)$ is true, nor he is sure that $P(x)$ is false. In this case, the expert's degree of confidence is *intermediate* between "true" and "false".

How can we represent these intermediate degrees of belief in a computer?

If 0 corresponds to "false", and 1 to "true", then it is natural to represent degrees of confidence that are intermediate between "false" and "true" by numbers from the interval $(0, 1)$.

Thus, to describe *arbitrary* degrees of certainty, we must describe:

- either absolute truth (represented by 1),
- or absolute falsity (represented by 0),
- or intermediate degrees of confidence (represented by numbers from the open interval $(0, 1)$).

Therefore, we arrive at the following conclusion:

We must use real numbers from the interval $[0, 1]$.

Historical comments.

- From the *mathematical* viewpoint, what we are doing is a very natural idea: we extend the traditional *2-valued* logic (in which the truth value of each statement is an element of a 2-element set $\{$"true", "false"$\} = \{0, 1\}$), to the *interval* $[0, 1]$. Mathematicians have been developing the corresponding mathematical formalisms (called *multiple-valued logics*) since the pioneer work of K. Lukasiewicz in the early 1920s.
- In *computer science*, the idea of using numbers from the interval $[0, 1]$ to describe different degrees of confidence was also first proposed by Lotfi Zadeh in his pioneer 1965 paper on fuzzy sets.

Comments.

- We have used the term *degree of confidence*; other words are also used, such as *degree of belief*, *degree of certainty*, *truth value*, *subjective probability*, etc. From our viewpoint, all of these terms are good but not perfect:
  - On one hand, each of these terms bring with itself some intuitive meaning.
  - On the other hand, the intuitive meaning brought by these words may be sometimes misleading. For example, the terms "belief" and "subjective probability" may lead to a confusion with the "objective probability", i.e., crudely speaking, with a *frequency* of a certain event (like tossing a coin).

  Since we cannot select one term as the best, we will use these terms interchangingly.

- The interval $[0, 1]$ is probably the most natural to use, but some expert systems use a *different interval* to represent uncertainty. For example, historically the first successful expert system MYCIN, developed at Stanford University for diagnosing rare blood diseases, used values from the interval $[-1, 1]$ to represent uncertainty.

- The main advantage of using an interval (e.g., $[0, 1]$) is that elements from this interval (i.e., real numbers) can be easily represented and easily processed in a computer. However, sometimes, to get a more adequate representation of the expert's degree of confidence, we may need a more sophisticated representation. For example, an expert may not be sure about his degree of confidence. To represent this uncertainty, we may want to describe the expert's degree of confidence not by a *single* number, but by an *interval* of possible numbers:
  - If an expert has no information about the object, he cannot have any definite degree of confidence. To describe this situation, we will use the entire interval $[0, 1]$ as the description of this expert's degree of confidence.
  - If an expert has a certain information, them we may use a narrower interval, or even a number (i.e., a degenerate interval).

  There are several even more sophisticated ways of representing uncertainty. In this text, we will only consider the simplest possible way of representing degrees of confidence: by using real numbers.

We need to elicit a numerical degree of confidence from the expert. We agreed that for each statement of the type $P(x)$, where:

- P is a fuzzy property (i.e., a property formulated by words from natural language), and
- x is a real number,

a reasonable way to represent the expert's degrees of confidence is to use numbers from the interval $[0, 1]$. The natural next question is: how can we "extract", elicit the value from an expert?

Direct elicitation is impossible. The ideal situation would be if an expert would directly provide us with this number, but this is, unfortunately, not realistic:

- we want numbers from the interval $[0, 1]$, because they are very *natural* for a *computer*; but
- real numbers from the interval $[0, 1]$ are *not natural* for a *human expert*; it is very difficult for most experts to express their degree of confidence in a given statement by a real number.

Since we cannot elicit these numbers *directly*, we have to use *indirect* elicitation techniques.

There are several dozen different elicitation techniques; in this lesson, we will only describe the most frequently used ones.

First elicitation method: selecting on a scale. If we cannot elicit a *real* number from an expert, maybe we can elicit *some* number from him, and then convert the result into a real number from the interval $[0, 1]$. In many cases, this is indeed possible: many polls ask us to estimate our degree of confidence, or our degree of belief, etc., on a certain integer scale, e.g., on a scale from 0 to 5, or on a scale from 0 to 10. So, we can ask an expert to mark his degree of confidence in a given statement $P(x)$ on a given scale 0 to S (0 to 5, 0 to 10, etc). On this scale:

- 0 corresponds to "absolutely confident that $P(x)$ is false";
- S corresponds to "absolutely confident that $P(x)$ is true";
- intermediate marks represent different degrees of uncertainty.

As a result of this procedure, we get an integer from 0 to S. The most natural way to convert these integers $0, 1, 2, \ldots, S$ into real numbers from the interval $[0, 1]$ is to *re-scale* the interval $[0, S]$ into the interval $[0, 1]$ by dividing by S. So, if an expert has chosen the mark s on a scale from 0 to S, we will take s/S as the desired degree of confidence.

For example, if an expert selects, 3 on a scale from 0 to 5, then we take $3/5 = 0.6$ as the desired degree of confidence.

The first elicitation method is not always applicable. The above method seems reasonable and is computationally very simple. The problem with this method is that experts often do not feel comfortable expressing their degree of confidence by numbers on an (integer) scale. Usually, these experts only distinguish between three cases:

- a given statement $P(x)$ is true;
- a given statement $P(x)$ is false;
- we do not know whether a given statement $P(x)$ is true or false.

If we have such experts, how can we elicit their degree of confidence?

Second elicitation method: polling. We have already mentioned that the first elicitation method is taken from the experience of the polls. In addition to the polls that require us to mark a point on a scale, there are other polls in which we only answer "yes", "no", or "undecided". For example, polls conducted before the elections are usually of this type. As a result of each poll of this type, we get a *percentage* of people who answered "yes" ("60% are planning to vote for candidate X"). This percentage represents exactly what we want: a number from the interval $[0, 1]$ (e.g., 60% means 0.6).

So, we arrive at the following *polling* method of eliciting the degree of confidence from the experts: we ask several experts whether $P(x)$ is true or false. Some of these experts may not answer at all, or give "unknown" as an answer; these experts we need not count. If out of N experts who gave a definite answer, M answered "yes" (i.e., "yes, $P(x)$ is true"), then we take the ratio M/N as the desired degree of confidence.

Problems with the second method. For the polling method to give meaningful results, we need many respondents. From the strict mathematical viewpoint, for a pool to give meaningful results (e.g., to give the percentage with a guaranteed 10% accuracy), we must interview at least 1,000 people.

- For an election poll, interviewing 1,000 people is quite possible: millions participate in the elections, and we want to know the opinion of the representative group of people.
- However, in an expert system, we are trying to formalize the knowledge of the top experts. We may not have that many top experts, and even if we do, top experts are usually very busy people, we may not be able to convince all of them to cooperate with this project. Last but not the least, the time of top experts costs money, and we may not have the money to hire 1,000 of them.

Can we somehow ask a single expert and still get a number?

Third elicitation method: using subjective probabilities and bets. Some people feel uncomfortable estimating a probability or marking a value on a scale, but they have a good feel for *bets.* Such people cannot express the probability of their favorite horse winning, but they are absolutely sure that, say, they are willing to bet 4 to 1 but not 5 to 1 that this horse will win.

If we can extract a betting ratio from an expert, then we can easily transform this number into the subjective probability, that will serve as the desired degree of confidence.

For example, if an expert is willing to bet 4 to 1 but not any better that $P(x)$ is true (e.g., that most people will agree that $x = 1$ is a small temperature difference), this means that this experts' subjective probability is $4/(4+1) = 0.8$.

From finitely many degrees of confidence to a membership function: extrapolation is needed. We started this section with a problem of describing a fuzzy property $P(x)$ (of the type "x is small"). Namely, for every possible value x, we would like to know the *degree of confidence* $d(P(x))$ that this value x

satisfies the property P. This degree of confidence is usually denoted by $\mu_P(x)$, and the function that transforms a real number x into a value $\mu_P(x) \in [0,1]$ is called a *membership function*, or a *fuzzy set*.

For each value x, we can, using one of the above elicitation techniques, find the value $\mu_P(x)$ for this x. However, this is not sufficient:

- On one hand, we want to know the value $\mu_P(x)$ for all possible real numbers x, i.e., for *infinitely many* different values.
- On the other hand, we can only ask an expert *finitely many* questions and therefore, we can only determine the values of the membership function for *finitely many* different values $x^{(1)}, \ldots, x^{(v)}$.

Thus, after all the elicitation is over, we only know the values $\mu_P(x^{(p)})$ of the desired function $\mu_P(x)$ for v different values $x^{(1)}, \ldots, x^{(v)}$. To reconstruct the desired function, we must use *extrapolation*.

Spline extrapolation is most frequently used. We have already argued in the previous lessons that if we have no information about x (and therefore, we have no reason to prefer a certain measuring unit for x), then it is reasonable to choose a class of extrapolation methods that is optimal with respect to a shift- and scale-invariant criterion, which means using a *piece-wise polynomial* (*spline*) extrapolation. This extrapolation is indeed most frequently used in control applications.

Other extrapolation methods and neuro-fuzzy control. Sometimes, more complicated extrapolation techniques such as neural networks lead to better control results. Such control in which neural networks are used together with fuzzy methods is called *neuro-fuzzy*.

Examples of extrapolation. Let us describe a few simple examples of extrapolation. The simplest extrapolation is piece-wise polynomial, and the simplest possible polynomial is a linear function. Thus, the simplest extrapolation is an extrapolation by *piece-wise linear functions*. The simplest case of extrapolation is when we start with the values x for which the expert is either absolutely sure that x is true, or he is absolutely sure that x is false, i.e., with values for which $\mu_P(x) = 0$ or $\mu_P(x) = 1$. Let us give several examples of such situations.

- Let us first describe the property of the type "x is negligible".

  – The only case when we are 100% sure that x is negligible is when $x = 0$. So, we have the value $\mu_P(0) = 1$.

  – Usually, we also know the value $\Delta > 0$ after which the difference in temperatures is no longer negligible. For example, for a thermostat that controls the room's temperature, we can take $\Delta = 10$. This means that $\mu_P(x) = 0$ for $x \geq \Delta$ and for $x < -\Delta$.

  We know the value of the function $\mu_P(x)$ for $x \leq -\Delta$, for $x = 0$, and for $x \geq \Delta$. We need to use linear extrapolation to find the values of this function for $x \in (-\Delta, 0)$ and for $x \in (0, \Delta)$. In general, the formula for a linear function $f(x)$ that passes through the point $y_1 = f(x_1)$ and $y_2 = f(x_2)$, is

$$f(x) = y_1 + (x - x_1) \cdot \frac{y_1 - y_2}{x_2 - x_1}.$$

  For our case, this formula leads to the following membership function:

  – To get the values $\mu_P(x)$ for $x \in (-\Delta, x)$, we take $x_1 = -\Delta$, $x_2 = 0$, $y_1 = 0$, and $y_2 = 1$. As a result, we get the expression $\mu_P(x) = (x + \Delta)/\Delta$.

  – To get the values $\mu_P(x)$ for $x \in (0, \Delta)$, we take $x_1 = 0$, $x_2 = \Delta$, $y_1 = 1$, and $y_2 = 0$. As a result, we get the expression $\mu_P(x) = 1 - x/\Delta$.

  Thus, the function $\mu_P(x)$ takes the following form:

  - $\mu_P(x) = 0$ for $x \leq -\Delta$;
  - $\mu_P(x) = (x + \Delta)/\Delta$ for $-\Delta \leq x \leq 0$;
  - $\mu_P(x) = 1 - x/\Delta$ for $0 \leq x \leq \Delta$; and
  - $\mu_P(x) = 0$ for $x > \Delta$.

  The graph of this function has the shape of a *triangle* over the x-axis. Therefore, such functions are called *triangular* membership functions.

  A similar shape describes properties like "close to a", for some fixed a. In this case, the triangular membership function has the form:

  - $\mu_P(x) = 0$ for $x \leq a - \Delta$;
  - $\mu_P(x) = (x - (a - \Delta))/\Delta$ for $a - \Delta \leq x \leq a$;
  - $\mu_P(x) = 1 - (x - a)/\Delta$ for $a \leq x \leq a + \Delta$; and
  - $\mu_P(x) = 0$ for $x > a + \Delta$.

- For the same property "x is negligible", in some cases, we know the lower bound δ below which the temperature difference x is indeed negligible. In this case, in addition to knowing that $\mu_P(x) = 0$ for $x \le -\Delta$ and for $x \ge \Delta$, we also know that $\mu_P(x) = 1$ for $-\delta \le x \le \delta$. For this function, linear extrapolation to the intervals $(-\Delta, -\delta)$ and (δ, Δ) results in the following membership function:

  - $\mu_P(x) = 0$ for $x \le -\Delta$;
  - $\mu_P(x) = (x + \Delta)/(\Delta - \delta)$ for $-\Delta \le x \le -\delta$;
  - $\mu_P(x) = 1$ for $-\delta \le x \le \delta$;
  - $\mu_P(x) = 1 - (x - \delta)/(\Delta - \delta)$ for $\delta \le x \le \Delta$; and
  - $\mu_P(x) = 0$ for $x > \Delta$.

  The graph of this function has the shape of a *trapezoid* over the x-axis. Therefore, such functions are called *trapezoidal* membership functions.

- Another frequent example of piece-wise functions are functions that describe properties like "x is large". For these properties, we usually know that values below a certain δ are definitely not large, and that values about a certain $\Delta \gg \delta$ are definitely large. So, we know that $\mu_P(x) = 0$ for $x \le \delta$ and $\mu_P(x) = 1$ for $x \ge \Delta$. To get the values of $\mu_P(x)$ for $x \in (\delta, \Delta)$, we use a linear extrapolation. As a result, we get the following function:

  - $\mu_P(x) = 0$ for $x \le \delta$;
  - $\mu_P(x) = (x - \delta)/(\Delta - \delta)$ for $\delta \le x \le \Delta$;
  - $\mu_P(x) = 1$ for $x \ge \Delta$.

We have described the triangular and trapezoid functions because they are *the simplest.* Interestingly, in fuzzy control, they are also, at present, *the most frequently used membership* functions. This fact has two possible explanations:

- First, fuzzy control is still at its infancy. The first successful application of fuzzy control was produced a little more than 20 years ago (in 1974, by E. Mamdani). Because of that, the potential of simple fuzzy control applications is not yet exhausted. *Maybe, we will need more complicated membership functions* later on, when all simple applications will be used, but so far, we do not seem to need them.

- Second, we are formalizing *approximate* expert knowledge. If all an expert can say about a control is that it should be *small*, or *medium*, or *large*, it may not be reasonable to try to formalize these notions in a too complicated way. Thus, simple models of expert uncertainty, in particular, simple membership functions, may be more reasonable than complicated ones. From this viewpoint, *we may not need more complicated membership functions at all.*

Probably, the truth lies somewhere in between; but anyway, so far, piecewise-linear functions seem to work just fine.

Exercises

13.1 Choose several values of temperature difference x and, using your friends as experts, apply one of the elicitation methods to describe the degrees of confidence that these values x are negligible.

13.2 Based on your elicitation results, use piecewise-linear extrapolation to form a membership function that corresponds to the word "negligible". Describe an analytical expression for this function, and plot its graph.

13.4. Representing natural-language properties: re-scaling

Different elicitation methods can lead to different results. The main reason why we had to describe *several* different elicitation methods is that each method has its limitations, and so:

- we need a *second* method to elicit degrees of confidence in the situation when the first method does not work;
- we need a *third* method to elicit degrees of confidence in the situation when the first two methods do not work;
- etc.

From this description, it is clear that usually, only one of these methods is applicable. However, there are situations when we can apply several different methods. The reader should be warned that in such situations, different methods can lead to somewhat different results. To be more precise:

- From the *qualitative* viewpoint, the values from different scales are usually consistent: if experts' confidence in a statement A is higher than the experts' confidence in the statement B, then for all reasonmable methods, the degree of belief $d(A)$ in A will be larger than the degree of belief in B: $d(A) > d(B)$.
- However, the actual *numerical*, *quantitative* values $d(A)$, $d(B)$ of these degrees of belief can differ from method to method.

The reason for this difference is that different methods may result in values corresponding to different *scales* of uncertainty, just like measuring the length in feet or in meters leads to different scales in which the order is preserved but numerical values are different. So, if we have different degrees of belief obtained by different methods, we may need to *re-scale* them to make them comparable.

Comment. As we have mentioned, in most applications of expert systems and fuzzy control, we do not encounter this problem. So, a reader who is only interested in learning about the very basics of fuzzy control, can skip this section.

What re-scaling should we choose? Working intelligent systems use several different procedures for assigning numeric values that describe uncertainty of the experts' statements. The same expert's degree of uncertainty that he expresses, for example, by the expression "for sure", can lead to 0.9 if we apply one procedure, and to 0.8 if another procedure is used. Just like 1 foot and 12 inches describe the same length, but in different scales, we can say that 0.9 and 0.8 represent the same degree of certainty in two different *scales*.

Some scales are different even in the fact that they use an interval different from [0,1] to represent uncertainty. For example, as we have already mentioned, the famous MYCIN system uses the interval $[-1, 1]$.

In some sense all scales are equal, but some are more reasonable than others. From a mathematical viewpoint, one can use *any* scale, but from the practical viewpoint some of them will be more reasonable to use, and some of them less reasonable. We'll consider only *practically reasonable* scales, and we'll try to formalize what that means.

We must describe transformations between the scales. Since we are not restricting ourselves to some specific procedure of assigning a numeric value to uncertainty, we can thus allow values from different scales. If we want to combine them, we must be able to transform them all to one scale. So we must be able to describe the transformations between reasonable scales (*"rescalings"*).

The class $\mathcal{R}$ of reasonable transformations of degrees of uncertainty must satisfy the following properties:

1) If a function $a \to r(a)$ is a reasonable transformation from a scale A to some scale B, and a function $b \to s(b)$ is a reasonable transformation from B into some other scale C, then it is reasonable to demand that the transformation $a \to s(r(a))$ from A to C is also a reasonable transformation. In other words, the class $\mathcal{R}$ of all reasonable transformations must be closed under composition.
2) If $a \to r(a)$ is a reasonable transformation from a scale A to scale B, then the inverse function is a reasonable transformation from B to A.

Thus, the family $\mathcal{R}$ must contain the inverse of every function that belongs to it, and the composition of every two functions from $\mathcal{R}$. In mathematical terms, it means that $\mathcal{R}$ must be a *transformation group*.

3) If the description of a rescaling is too long, it is unnatural to call it reasonable. Therefore, we will assume that the elements of $\mathcal{R}$ can be described by fixing the values of p parameters (for some small p).

In mathematics, the notion of a group whose elements are continuously depending on finitely many parameters is formalized as the notion of a (connected) *Lie group*. So we conclude that reasonable rescalings form a connected transformation Lie group.

4) The last natural demand that we'll use is as follows. Of course, in principle, it is possible that we assign 0.1 in one scale and it corresponds to 0.3 in another scale. It is also possible that we have 0.1 and 0.9 on one scale that comprises only the statements with low degrees of belief, and when we turn to some other scale that takes all possible degrees of belief into consideration, we get small numbers for both. But if in some scale we have

the values 0.5, 0.51 and 0.99, meaning that our degrees of belief in the first two statements almost coincide, then it is difficult to imagine another reasonable scale in which the same three statements have equidistant truth values, say 0.1, 0.5 and 0.9. If this example is not convincing, take 0.501 or 0.5001 for the second value on the initial scale. We'll formulate this idea in the maximally flexible form: *there exist two triples of truth value that cannot be transformed into each other by any natural rescaling.*

Examples of reasonable rescaling transformations. In addition to these general demands, we have some examples of rescalings that are evidently reasonable.

Indeed, one of the natural methods to ascribe the degree of confidence $d(A)$ to a statement A is to take several (N) experts, and ask each of them whether she believes that A is true. If $N(A)$ of them answer "yes", we take $d(A) = N(A)/N$ as the desired certainty value. (This method is a slight modification of the polling method, a modification in which we do not dismiss those whose do not answer anything definite.) If all the experts believe in A, then this value is 1 (=100%), if half of them believe in A, then $t(A) = 0.5$ (50%), etc.

Knowledge engineers want the system to include the knowledge of the entire scientific community, so they ask as many experts as possible. But asking too many experts leads to the following negative phenomenon: when the opinion of the most respected professors, Nobel-prize winners, etc., is known, some less self-confident experts will not be brave enough to express their own opinions, so they will either say nothing or follow the opinion of the majority. How does their presence influence the resulting uncertainty value?

- Let N denote the initial number of experts, $N(A)$ the number of those of them who believe in A, and M the number of shy experts added. Initially, $d(A) = N(A)/N$. After we add M experts who do not answer anything when asked about A, the number of experts who believe in A is still $N(A)$, but the total number of experts is bigger $(M + N)$. So the new value of the uncertainty ratio is

  $$d'(A) = \frac{N(A)}{N + M} = c \cdot d(A),$$

  where we denoted $c = N/(M + N)$.

- When we add experts who give the same answers as the majority of N renowned experts, then, for the case when $d(A) > 1/2$, we get $N(A) + M$

experts saying that A is true, so the new uncertainty value is

$$d'(A) = \frac{N(A) + M}{N + M} = \frac{N \cdot d(A) + M}{N + M}.$$

- If we add M "silent" experts and M' "conformists" (who vote as the majority), then we get a transformation

$$d(A) \to \frac{N \cdot d(A) + M'}{N + M + M'}.$$

In all these cases the transformation from an old scale $d(A)$ to a new scale $d'(A)$ is a linear function $d(A) \to a \cdot d(A)b$ for some constants a and b; in the most general case $a = N/(N + M + M')$ and $b = M'/(N + M + M')$. By selecting appropriate values of N, M, and M', we can get arbitrary linear functions with linear coefficients. Thus, we arrive at the following conclusion:

The problem of selecting re-scalings re-formulated in mathematical terms.

> *The set of possible re-scalings form a Lie transformation group that contains all linear transformations.*

This re-formulation leads to a solution.

Description of all possible re-scaling for the case when the property 4) is not required. We already know, from Lesson 11, that each such transformation is fractionally linear. Thus,

> *Every reasonable rescaling of degrees of confidence can be described by a fractionally linear function.*

Description of all possible re-scaling for the case when the property 4) is required. Let us show that if we require property 4), then only linear transformations are possible.

Indeed, to be more precise, the desired Lie group either consists of only linear transformations, or of all fractionally linear ones. In the second case, we get a violation of the property 4): indeed, if we have three degrees $d_1 < d_2 < d_3$, and we want to transform them into three other degrees $d_1' < d_2' < d_3'$ by using an appropriate fractionally-linear transformation

$$d \to d' = \frac{a \cdot d + b}{1 + c \cdot d}$$

for appropriate a, b, and c, then we must find these coefficients a, b, and c from the following system of three equations with three unknowns:

$$d_i' = \frac{a \cdot d_i + b}{1 + c \cdot d_i} \; (i = 1, 2, 3)$$

Multiplying both sides of each equation by its denominator, we get a system of three linear equations with three unknowns:

$$d_i' + c \cdot d_i \cdot d_i' = a \cdot d_i + b, \; (i = 1, 2, 3)$$

From this linear system, we can easily find a, b, and c. Thus, if we require the property 4), we arrive at the following conclusion:

Every reasonable rescaling of degrees of confidence can be described by a linear function.

Exercise

13.3 Find a fractionally linear transformation $d \to d'$ that transforms the values $d_1 = 0.5$, $d_2 = 0.51$, and $d_3 = 0.99$ into $d_1' = 0.1$, $d_2' = 0.5$, and $d_3' = 0.9$.

14
INTELLIGENT AND FUZZY CONTROL

In this lesson, we continue the description of mathematical methods for dealing with expert knowledge; namely, we describe, step-by-step, the fuzzy control methodology that transforms the control rules (that expert operators formulate by using words of a natural language) into a precise control strategy.

14.1. Representing logical connectives

How can we represent logical connectives? An ideal solution. Suppose that an expert system contains statements A and B, and we have elicited the degrees of belief $d(A)$ and $d(B)$ from the experts. Suppose now that a user wants to know the degree of belief in a composite statement $A\&B$.

In principle, knowing only the two numbers $d(A)$ and $d(B)$ is not sufficient to describe the expert's degree of belief in $A\&B$: e.g., if $d(A) = d(B)$,

- it could be that the experts perceive A and B as *equivalent*, in which case $d(A\&B) = d(A)$, or
- it could also be that the experts perceive A and B as *independent* statements, in which case the possibility of A *and* B being true is smaller than the possibility that one of them is true: $d(A\&B) < d(A)$.

So, the *ideal* situation would be to elicit, from the experts, not only the degree of belief in the *basic* statements from the knowledge base, but also the degrees of belief in all possible *logical combinations* of these statements.

This ideal solution is not practically possible. The above described ideal solution is practically impossible: If we have N statements $S_1, ..., S_N$ in the knowledge base, then for each of $2^N - 1$ non-empty subsets $\{S_{i_1}, ..., S_{i_k}\}$ of the knowledge base, we need to elicit the degree of belief in the corresponding AND-statement $S_{i_1} \& ... \& S_{i_k}$. For a realistic expert system, N is in hundreds, so asking an expert 2^N questions is impossible.

A practical way to represent logical connectives: logical operations. In view of this practical impossibility, although in *some* cases, we will be able to have the degree of belief $d(A\&B)$ stored in the knowledge base, in *general*, we often have to deal with a following situation:

- we know the degrees of belief $d(A)$ and $d(B)$ in statements A and B;
- we know nothing else about A and B; and
- we are interested in the (estimated) degree of belief of the composite statement $A\&B$.

Since the only information available consists of the values $d(A)$ and $d(B)$, we must compute $d(A\&B)$ based on these values. We must be able to do that for arbitrary values $d(A)$ and $d(B)$. Therefore, we need a *function* that transforms the values $d(A)$ and $d(B)$ into an estimate for $d(A\&B)$. Such a function is called an *AND-operation*. If an AND-operation $f_\& : [0,1] \times [0,1] \to [0,1]$ is fixed, then we take $f_\&(d(A), d(B))$ as an estimate for $d(A\&B)$.

Similarly:

- to estimate the degree of belief in $A \vee B$, we need an *OR-operation* $f_\vee : [0,1] \times [0,1] \to [0,1]$.
- to estimate the degree of belief in the negation $\neg A$, we need a *NOT-operation* $f_\vee : [0,1] \times [0,1]$.

Terminological comment. AND operations are also called *t-norms*, and OR operations are also called *t-conorms.*

Natural properties of logical operations. The logical operations with fuzzy values must satisfy some *natural conditions.*

- For an expert, $A\&B$ and $B\&A$ mean the same. Therefore, the estimates $f_\&(d(A), d(B))$ and $f_\&(d(B), d(A))$ for these two statements should coincide. To achieve that, we must require that $f_\&(a, b) = f_\&(b, a)$ for all a and b; in other words, the operation $f_\&$ must be *commutative*.
- Similarly, from the fact that $A\&(B\&C)$ and $(A\&B)\&C$ mean the same, we can deduce the requirement that $f_\&$ must be *associative*: $f_\&(a, f_\&(b, c)) = f_\&(f_\&(a, b), c)$ for all a, b, and c.
- If A is absolutely false ($d(A) = 0$), then $A\&B$ is also absolutely false, i.e., $f_\&(a, 0) = 0$ for all a.
- If A is absolutely true ($d(A) = 1$), then $A\&B$ is true iff B is true, so, the degree of belief in $A\&B$ must coincide with the degree of belief in B: $f_\&(1, b) = b$ for all b.
- If our belief in A or B increases, then the degree of belief in $A\&B$ must also increase, so $f_\&$ must be *monotonic*.
- If the degree of belief in A changes a little bit, then the degree of belief in $A\&B$ must also change slightly. In other words, $f_\&$ must be *continuous*.

Combining all these requirements, we arrive at the following definition:

Definition 14.1.

- *By an AND-operation, we mean a commutative, associate, monotonic, continuous operation* $f_\& : [0, 1]\times[0, 1] \to [0, 1]$ *with the properties* $f_\&(1, a) = a$ *and* $f_\&(0, a) = 0$.
- *By an OR operation, we mean a commutative, associate, monotonic, continuous operation* $f_\vee : [0, 1]\times[0, 1] \to [0, 1]$ *with the properties* $f_\vee(1, a) = 1$ *and* $f_\vee(0, a) = a$.

How can we determine AND and OR operations? Since our goal is to describe the expert knowledge, we must elicit these operations from the experts. This can be done in the following manner:

- We form several pairs of statements (A_k, B_k), $k = 1, 2, \dots$
- For each pair from this set, we elicit, from the experts, the degrees of confidence $d(A_k)$, $d(B_k)$, and $d(A_k\&B_k)$.

- Then, we use an extrapolation procedure to find a function $f_\&(a, b)$ for which $f_\&(d(A_k), d(B_k)) \approx d(A_k \& B_k)$.

Similar procedures enable us to determine OR and NOT operations.

Historical comment. This empirical approach was first implemented and used by the designers of the first successful expert system MYCIN. After spending several years, they came up with AND and OR operations that fit the reasoning of medical experts. Interestingly, it turned out that different medical experts use very similar AND and OR operations. This lead MYCIN designers to a natural hypothesis that these operations are indeed general for human reasoning. Alas, when they tried to apply these same AND and OR operations to another area (geology), the resulting expert system did not lead to good results. It turned out that in different fields, people use different AND and OR operations. This difference is easy to explain:

- In some applications (e.g., in *medicine*), mistakes can be deadly, so more *cautious* estimates are needed (e.g., $f_\&(a, b) = a \cdot b$).
- In other applications (e.g., in *geology*), we cannot measure as many parameters as in medicine, so, we have to rely more on expertise, and hence, experts must take risks. In these applications, more brave, more optimistic estimates are needed: e.g., a geologist starts to drill in the uncertainty in which a surgeon is not likely to start an incision. Therefore, for such applications, we need more *optimistic* estimates for $d(A\&B)$ (e.g., $f_\&(a, b) = \min(a, b)$).

Simple AND, OR, and NOT operations: an idea. There are many possible AND and OR operations, many of them very complicated. However, in most applications, we do not need this complexity:

- Our goal is to apply these operations to the degrees of confidence, that, in their turn, come from the values of membership functions.
- We have already mentioned that in most applications, it is sufficient to use the *simplest* membership functions, that are obtained by applying the *simplest* spline interpolation methods to crisp values of the corresponding properties.

Since we start with the simplified expressions for degrees of confidence, it makes sense to consider *simple* AND and OR operations for handling these degrees of confidence, i.e., to consider AND and OR operations that are obtained by applying some *simple spline* extrapolation techniques (ideally, linear ones) to the *crisp* values of these operations.

Linear NOT operation. A NOT operation $f_\neg(a)$ must, given the degree of belief $d(A)$ in a statement A, return the degree of belief $f_\neg(d(A))$ in its negation $\neg A$. For crisp values $d(A) = 0, 1$, we have $f_\neg(0) = 1$ and $f_\neg(1) = 0$. One can easily show that for NOT, we can actually have a linear operation, and this linear NOT operation is uniquely defined:

Definition 14.2. *We say that a function $f_\neg(a)$ from $[0,1]$ to $[0,1]$ is a linear NOT operation if $f_\neg(a)$ is a linear function for which $f_\neg(0) = 1$ and $f_\neg(1) = 0$.*

Proposition 14.1. *$f_\neg(a) = 1 - a$ is the only linear NOT operation.*

Comments.

- The proof is straightforward: for every two points (x_1, y_1) and (x_2, y_2), there exists one and only one linear function

$$f(x) = y_1 + (x - x_1) \cdot \frac{y_2 - y_1}{x_2 - x_1}$$

  for which $f(x_1) = y_1$ and $f(x_2) = y_2$. In our case, $x_1 = 0$, $x_2 = 1$, $y_1 = 0$, and $y_2 = 1$.

- The NOT operation $f_\neg(a) = 1 - a$ is indeed most frequently used in fuzzy logic.

- The resulting formula $d(\neg A) = f_\neg(d(A)) = 1 - d(A)$ for the *degree of belief* in $\neg A$ resembles a formula for the *probability* of $\neg A$: If we know the probability $P(A)$ of an event A, then the probability $P(\neg A)$ that this event will not occur is equal to $P(\neg A) = 1 - P(A)$. Thus, the linear NOT operation is consistent with the two probability-like elicitation procedures for degree of belief: namely, with the procedures that are based on polling and betting.

There exist no linear AND and OR operations. For NOT, we have found a linear operation. For AND and OR, the situation is more complicated because there are no linear AND and OR operations:

Definition 14.3.

- *We say that a function $f_\&(a,b)$ from $[0,1]\times[0,1]$ to $[0,1]$ is a linear AND operation if $f_\&(a,b)$ is a linear function for which $f_\&(0,0)=f_\&(0,1)=f_\&(1,0)=0$ and $f_\&(1,1)=1$.*
- *We say that a function $f_\vee(a,b)$ from $[0,1]\times[0,1]$ to $[0,1]$ is a linear OR operation if $f_\vee(a,b)$ is a linear function for which $f_\vee(0,0)=0$ and $f_\vee(0,1)=f_\vee(1,0)=f_\&(1,1)=1$.*

Proposition 14.2.

- *There exist no linear AND operations.*
- *There exist no linear OR operations.*

Comment. In the remaining part of this section, we will only prove the results about AND operations. Corresponding results about OR operations can be proved in a similar manner; these proofs are left for the readers.

Proof. In accordance with what we have just mentioned, we will only prove Proposition 14.2 for AND operations. A general linear function of two variables has the form $f_\&(a,b)=c_0+c_a\cdot a+c_b\cdot b$ for some coefficients c_0, c_a, and c_b. If we substitute this expression into the four conditions that describe a linear AND operation, we will get a system of four equations for three unknowns: $c_0=0$, $c_0+c_b=0$, $c_0+c_a=0$, and $c_0+c_a+c_b=1$. From the first three equations, we can uniquely determine the three coefficients c_0, c_a, and c_b:

- from the first equation, we conclude that $c_0=0$, and then
- from the second and third equations we conclude that $c_a=c_b=0$.

Alas, if we we substitute these values into the fourth equation, we get $c_0+c_a+c_b=0\neq 1$. Thus, no linear function satisfies all four equations, and hence, linear AND operations are impossible. Proposition is proven.

Comments.

- This result is well known to people familiar with the history of *neural networks*: Since neural networks have been proposed as *universal* computing devices, capable of doing both numerical computations and logical reasoning, researchers tried to implement logical operations (such as AND and OR) in neural networks. For quite some time, these attempts were un-successful, until researchers realized that they were trying to use *linear* neurons that can only implement *linear* operations, and with linear operations, we cannot implement AND or OR. Thus, the impossibility of linear AND and OR operations was one of the main motivations for *non-linear* neurons.

- We have decided to use splines (piece-wise polynomial functions) to represent logical operations. Since we cannot use the *simplest* splines (i.e., linear functions), we must take the *next simplest*. A linear function is the simplest case of a spline, in which:

  - there is only *one* piece; and
  - the corresponding polynomial is of the smallest possible degree (*one*).

  To get the next simplest case, we must relax one of these two conditions. Thus, we have two options:

  - One option is to still consider splines formed by polynomials of the smallest degree (one), but allow *two* pieces instead of one. In this case, we get *piecewise-linear* functions, that consist of *two* different linear functions on two parts of the square $[0,1] \times [0,1]$.
  - Another option is to still consider functions consisting of only one piece, but to allow a higher degree (two). In this case, we get *quadratic* functions.

  We will see that in both cases, we get reasonable AND and OR operations.

Piecewise-linear AND and OR operations. We want an AND operation to satisfy four conditions that correspond to $a = 0, 1$ and $b = 0, 1$. Geometrically, if we describe the set of all possible values (a, b) as a square, these four conditions correspond to the *vertices* of this square.

A general linear function of two variables a and b can be described by three coefficients c_0, c_a, and c_b. Initially, we tried to find a linear function that satisfies the four conditions that correspond to all four vertices of the square. As a result, we got an “over-determined” system of four equations with three unknowns. Since we cannot have a linear function that fits all four vertices of the square, we must divide this square into pieces on which a fit is possible.

Each vertex leads to a linear equation for the coefficients c_0, c_a, and c_b. To determine all three coefficients, we must have three equations for each piece. Thus, each of the two pieces must be determined by three vertices. So, from the geometric viewpoint, to find an appropriate piecewise-linear function, we must:

- group the vertices of the square into two groups of three so that the square will be divided into the two triangles, and
- on each of these triangles, determine the corresponding linear function.

There are only two ways to divide the square into two triangles:

- We can divide the square by a diagonal that goes from the lower left vertex $(0,0)$ to upper right vertex $(1,1)$. In this case:
  - the first triangle has the vertices $(0,0)$, $(1,0)$, and $(1,1)$;
  - the second triangle has the vertices $(0,0)$, $(0,1)$, and $(1,1)$.
- We can also divide the square by a diagonal that goes from upper left vertex $(0,1)$ to the lower right vertex $(1,0)$. In this case:
  - the first triangle has the vertices $(0,0)$, $(0,1)$, and $(1,0)$;
  - the second triangle has the vertices $(0,1)$, $(1,0)$, and $(1,1)$.

In both cases, we can easily determine the coefficients of each of the two linear pieces:

- For the first division, we get $f_\&(a,b) = b$ for the first triangle and $f_\&(a,b) = a$ for the second triangle. These two formulas can be combined into a single expression $f_\&(a,b) = \min(a,b)$.
- For the second division, we get $f_\&(a,b) = 0$ for the first triangle and $f_\&(a,b) = a + b - 1$ for the second triangle. These two formulas can be combined into a single expression $f_\&(a,b) = \max(0, a+b-1)$.

As a result, we get two possible piece-wise linear AND operations: $f_\&(a,b) = \min(a,b)$ and $f_\&(a,b) = \max(0, a+b-1)$. Similarly, we get two possible OR operations: $f_\vee(a,b) = \max(a,b)$ and $f_\vee(a,b) = \min(a+b,1)$.

Comments.

- In knowledge representation, the operations $\min(a, b)$ and $\max(a, b)$ were first proposed in the pioneer 1965 paper of L. Zadeh. Computationally, they are the simplest. The operations $\max(0, a + b - 1)$ and $\min(a + b, 1)$ were used by R. Giles in 1976 under the name of *bold* operations.
- Similarly to the linear NOT operation, these four operations are also consistent with the probability-like interpretation: namely, if we know the probabilities $a = P(A)$ and $b = P(B)$ of two events A and B, then:
  - The probability $P(A\&B)$ that both events A and B will occur can take any real value from the interval $[\max(a + b - 1, 0), \min(a, b)]$.
  - The probability $P(A \vee B)$ that at least one of the events A or B will occur can take any real value from the interval $[\max(a, b), \min(a + b, 1)]$.

Quadratic AND and OR operations. In addition to *piece-wise linear* operations, another option is to use *quadratic* operations. A general quadratic function $f(a, b)$ of two variables has the form

$$f(a, b) = c_0 + c_a \cdot a + c_b \cdot b + c_{aa} \cdot a^2 + c_{ab} \cdot a \cdot b + c_{bb} \cdot b^2$$

with six coefficients c_0, c_a, c_b, c_{aa}, c_{ab}, and c_{bb}.

The four conditions on AND or OR operations, conditions that describe the values of these operations for crisp a and b (i.e., for $a, b \in \{0, 1\}$), lead to only four linear equations. Four linear equations cannot uniquely determine the values of six unknowns. Thus, in contrast to the piecewise-linear case, these four conditions are not sufficient to determine a quadratic function. We will show, however, that if we add a natural requirement of *monotonicity*, we will be able to determine these operations uniquely.

Definition 14.4. *We say that a function $f(a, b)$ is*

- *non-decreasing in a if for every a, a', and b, $a < a'$ implies $f(a, b) \le f(a', b)$.*
- *non-decreasing in b if for every a, b, and b', $b < b'$ implies $f(a, b) \le f(a, b')$.*
- *non-decreasing in each argument if it is both non-decreasing in a and non-decreasing in b.*

Definition 14.5.

- *We say that a function $f_\&(a,b)$ from $[0,1]\times[0,1]$ to $[0,1]$ is a quadratic AND operation if $f_\&(a,b)$ is a quadratic function that is non-decreasing in each argument and for which $f_\&(0,0)=f_\&(0,1)=f_\&(1,0)=0$ and $f_\&(1,1)=1$.*
- *We say that a function $f_\vee(a,b)$ from $[0,1]\times[0,1]$ to $[0,1]$ is a quadratic OR operation if $f_\vee(a,b)$ is a quadratic function that is non-decreasing in each argument and for which $f_\vee(0,0)=0$ and $f_\vee(0,1)=f_\vee(1,0)=f_\&(1,1)=1$.*

Proposition 14.3.

- *$f_\&(a,b)=a\cdot b$ is the only quadratic AND operation.*
- *$f_\vee(a,b)=a+b-a\cdot b$ is the only quadratic OR operation.*

Proof. Similarly to the previous proposition, we will only prove the result for AND operations; for OR operations, the proof is similar and this proof is left to the reader.

- From the condition $f_\&(0,0)=0$, we conclude that $c_0=0$.
- Now, from the condition $f_\&(0,1)=0$, we conclude that $c_b+c_{bb}=0$. Therefore, $c_{bb}=-c_b$.
- Similarly, from the condition that $f_\&(1,0)=0$, we conclude that $c_{aa}=-c_a$. Hence,
$$f_\&(a,b)=c_a\cdot(a-a^2)+c_b\cdot(b-b^2)+c_{ab}\cdot a\cdot b.$$
- From the condition $f_\&(1,1)=1$, we conclude that $c_{ab}=1$, and hence, $f_\&(a,b)=c_a\cdot(a-a^2)+c_b\cdot(b-b^2)+a\cdot b$.
- The function $f_\&(a,b)$ must take non-negative values for all a and b from the interval $[0,1]$. In particular, for $a=0$ and $b=0.5$, we get $f_\&(0,0.5)=c_b\cdot 0.25\ge 0$, hence, $c_b\ge 0$. Similarly, from the condition $f_\&(0.5,0)\ge 0$, we conclude that $c_a\ge 0$.

- Since the function $f_\&(a,b)$ is non-decreasing in a, we conclude, for $a = 0.5$, $a' = 1$, and $b = 0$, that $f_\&(0.5,0) = c_b \cdot 0.25 \le f_\&(1,0) = 0$, i.e., that $c_a \le 0$. Since we already know that $c_a \ge 0$, we conclude that $c_a = 0$.
- Similarly, we can conclude that $c_b = 0$ and therefore, that $f_\&(a,b) = a \cdot b$.

The proposition is proven.

Comments.

- In knowledge representation, the operations $a \cdot b$ and $a + b - a \cdot b$ were first proposed in the pioneer 1965 paper of L. Zadeh. They are called *algebraic product* and *algebraic sum*.
- Similarly to the linear NOT operation and to the piece-wise linear AND and OR operations, the two quadratic AND and OR operations are also consistent with the probability-like interpretation: namely, if we know the probabilities $a = P(A)$ and $b = P(B)$ of two events A and B, and we know that the events A and B are *independent*, then:
  - The probability $P(A \& B)$ that both events A and B will occur is equal to $a \cdot b = P(A) \cdot P(B)$.
  - The probability $P(A \vee B)$ that at least one of the events A or B will occur is equal to $a + b - a \cdot b = P(A) + P(B) - P(A) \cdot P(B)$.

More complicated AND and OR operations. The above-described six AND and OR operations are the ones that are most frequently used in expert systems and in fuzzy control. However, these operations are not the only possible ones. Indeed, as we have mentioned in Section 13.4, we can *re-scale* the scale of degrees of belief, i.e., we represent the degree of confidence $a = d(A)$ in the statement A not by a value a, but by a new value $a' = d'(A) = r(a) = r(d(A))$ for some monotonic function $r(a)$.

In Section 13.4, we considered *reasonable* rescalings that correspond to difference ways of assigning the degree of confidence, but we may also have other re-scalings that are motivated by indirect reasons such as computational simplicity.

For example, if we are only interested in applying the AND operation, and we have chosen the quadratic AND operation $f_\&(a,b) = a \cdot b$, then we can simplify computations by using a *logarithmic scale* ($a' = r(a) = \ln(a)$), in which the AND operation is much faster: indeed,

$$d'(A\&B) = \ln(d(A\&B)) = \ln(d(A) \cdot d(B)) =$$

$$\ln(d(A)) + \ln(d(B)) = d'(A) + d'(B),$$

so in the new scale, we need addition instead of multiplication, and on the computers, addition is usually much faster.

This example shows that it makes sense to consider arbitrary monotonic re-scaling functions $r(a)$.

How will an AND operation that has the form $c = f_\&(a, b)$ in the old scale look in the new scale $a' = r(a)$? In other words, if we know the degrees of confidence $a' = d'(A) = r(d(A))$ and $b' = d'(B) = r(d(B))$ in the new scale, what will be the degree of belief $c' = d'(A\&B) = r(A\&B))$ in this new scale? To get the expression for c' in terms of a' and b', we must do the following:

- First, we convert the values a' and b' into the old scale by applying the inverse re-scaling r^{-1}: $a = r^{-1}(a')$ and $b = r^{-1}(b')$.
- Then, we apply the AND operation $f_\&(a, b)$ to the values a and b expressed in the old scale. As a result, we get the value $c = f_\&(a, b) = f_\&(r^{-1}(a'), r^{-1}(b'))$.
- Finally, we convert the value c into the new scale by applying the re-scaling $r(a)$: $c' = r(c)$.

As a result of this three-step procedure, we get a new AND operation

$$f'_\&(a', b') = r(f_\&(r^{-1}(a'), r^{-1}(b'))).$$

Similarly, we get a new OR operation

$$f'_\vee(a', b') = r(f_\vee(r^{-1}(a'), r^{-1}(b')))$$

and a new NOT operation

$$f'_\neg(a') = r(f_\neg(r^{-1}(a'))).$$

The new operations are called *isomorphic* to the old ones, because they represent the same operations but on a different scale.

There are special names for operations that are isomorphic to piece-wise linear and quadratic AND and OR operations. (These names may sound somewhat strange for a computer science reader, because they were invented before the computer applications and they describe algebraic properties of the corresponding operations.)

- Operations that are isomorphic to quadratic AND and OR operations, i.e., operations of the type

  $$f'_\&(a', b') = r(r^{-1}(a') \cdot r^{-1}(b'))$$

  and

  $$f'_\vee(a', b') = r(r^{-1}(a') + r^{-1}(b') - r^{-1}(a') \cdot r^{-1}(b')),$$

  for some strictly increasing continuous function $r(a)$, are called *strictly Archimedean* AND and OR operations.

- Operations that are isomorphic to bold AND and OR, i.e., operations of the type

  $$f'_\&(a', b') = r(\max(r^{-1}(a') + r^{-1}(b') - 1, 0))$$

  and

  $$f'_\vee(a', b') = r(\min(r^{-1}(a') + r^{-1}(b'), 1)),$$

  for some strictly increasing continuous function $r(a)$, are called *non-strictly Archimedean* AND and OR operations.

- If we use $f_\&(a, b) = \min(a, b)$ and $f_\vee(a, b) = \max(a, b)$, then, as one can see, for every strictly monotonic function $r(a)$, the isomorphic operations $f'_\&(a', b')$ and $f'_\vee(a', b')$ have exactly the same form: $f'_\&(a', b') = \min(a', b')$ and $f'_\vee(a', b') = \max(a', b')$. These two operations are called *idempotent.*

Most of the AND and OR operations that are actually used belong to one of these three types.

In addition of these three classes of operations, we may consider even more complicated AND and OR operations that are, e.g., isomorphic to an idempotent operation on one subinterval of the interval $[0, 1]$ and to a strictly Archimedean one on another subinterval of this interval $[0, 1]$. It turns out that this combination covers *all* possible AND and OR operations: namely, for an arbitrary AND and OR operation that satisfies several reasonable properties (e.g., the ones described in Definition 14.1), we can sub-divide the interval $[0, 1]$ into sub-intervals on each of which the operation is isomorphic to an operation from one of the three classes. This general classification result was proven by Ling in 1965; see the textbooks by Klir–Yuan and Nguyen–Walker (mentioned in Lesson 13) for details.

Exercise

14.1 Prove the above results about OR operations.

14.2. Representing rules

Problems with implication. We started the description of the expert knowledge by mentioning that the expert knowledge is usually represented by "if-then" rules. Since we want to formalize these rules, it seems reasonable to formalize the statements of the type "if A then B", i.e., using a logical term for such statements, to formalize *implication* in the same way as we formalized "and", "or", and "not" operations. There are, however, two problems with this idea:

- First, a *minor* problem: unlike "and", "or", and "not", implication is usually *not directly implemented* in the computers. Our goal is to describe the expert knowledge for a computer. Most computer languages have built-in logical operations "and", "or", and "not", but usually, not implication. Therefore, even if all the statements are crisp, when we formalize these statements for the computer, we will still need to first *reformulate* implication in terms of other logical operations.
- Second, a *major* problem: implication is somewhat *counter-intuitive*. Namely, researchers working in mathematical logic are using a formalization of implication in which implication is defined by a truth table in which $A \to B$ is true in all cases except when A is true and B is false. So, if A is false, then $A \to B$ is true for an arbitrary statement B. For example, we can conclude that "if $2+2=5$, then witches have six wings". Such statement may be mathematically correct, but they are absolutely *counter-intuitive*, because our intuitive understanding of "if-then" assumes some relation, while in the witches example, there is no relation whatsoever between the assumption (that $2+2=5$) and the conclusion (that witches have six wings).

Due to these problems, most expert systems and intelligent control systems do not *directly* formalize implication but instead, try first to *reformulate* if-then rules in terms of "and", "or", and "not". How can we do that?

Reformulating if-then rules in terms of "and", "or", and "not": example. Let's first consider our toy thermostat example, with three rules (13.1)–(13.3). If we know the difference x between the actual and the desired temperature, what control u should we apply? We have three rules that describe when a control is reasonable. Therefore, u is a reasonable control if one of the the three rules is applicable, i.e., when either:

- the first rule is applicable (i.e., x is negligible) and u is negligible; or
- the second rule is applicable (i.e., x is small positive), and u is small negative; or
- the third rule is applicable (i.e., x is small negative), and u is small positive.

Summarizing, we can say that u is an appropriate choice for a control if and only if either (x is negligible *and* u is negligible), *or* (x is small positive *and* u is small negative), etc.

Let us describe this statement in more succinct terms. Let us use the following notations:

- $R_k(x, u)$ will indicate that k-th rule is applicable for a given x, and that this rule recommends to use the control value u;
- $C(x, u)$ will indicate that u is a reasonable control for a given input x.

Then, in the above example, we get

$$C(x, u) \equiv R_1(x, u) \vee R_2(x, u) \vee R_2(x, u),$$

where

$$R_1(x, u) \equiv N(x)\&N(u); \quad R_2(x, u) \equiv SP(x)\&SN(u);$$

$$R_3(x, u) \equiv SN(x)\&SP(u).$$

We already know how to formalize the properties (as membership functions, or fuzzy sets), and we know how to formalize the "and" and "or" operations.

Thus, for every input x, we can define the degree of belief $d_r(x, u)$ that r-th rule will be fired:

$$d_1(x, u) = f_\&(\mu_N(x), \mu_N(u)); \quad d_2(x, u) = f_\&(\mu_{SP}(x), \mu_{SN}(u));$$
$$d_3(x, u) = f_\&(\mu_{SN}(x), \mu_{SP}(u)).$$

Reformulating if-then rules in terms of "and", "or", and "not": general case. For a general rule base with R rules of the type with

$$A_{r1}(x_1)\& \ldots \& A_{rn}(x_n) \rightarrow B_r(y), \tag{13.4}$$

we get

$$C(x_1, \ldots, x_n, y) \equiv R_1(x_1, \ldots, x_n, y) \vee \ldots \vee R_R(x_1, \ldots, x_n, y), \tag{14.1}$$

where

$$R_r(x_1, \ldots, x_n, y) \equiv A_{r1}(x_1)\& \ldots \& A_{rn}(x_n)\& B_r(y). \tag{14.2}$$

Therefore, for each rule, the "firing degree" is equal to:

$$d_r(x_1, \ldots, x_n, y) = f_\&(\mu_{r1}(x_1), \ldots, \mu_{rn}(x_n), \mu_r(y)), \tag{14.3}$$

where:

- $\mu_{ri}(x_i)$ is the membership function corresponding to the property A_{ri};
- $\mu_r(y)$ is the membership function corresponding to the property $B_r(u)$;
- $f_\&(a, b, c)$ stands for $f_\&(f_\&(a, b), c)$, and, in general, $f_\&(a_1, \ldots, a_n, a_{n+1}) = f_\&(f_\&(a_1, \ldots, a_n), a_{n+1})$.

14.3. Representing rule bases

General idea. In the previous section, we have shown how to transform the formula (14.2) into an algorithm that describes the degree of belief $d_r(x_1, \ldots, x_n, y)$ that each rule is applicable. After we have computed the firing degree of each rule, we can similarly formalize the formula (14.1) and get a numerical value that describes to what extent each possible control value y is reasonable for a given input $x_1, \ldots, x_n$:

$$\mu_C(x_1, \ldots, x_n, y) = f_\vee(d_1(x_1, \ldots, x_n, y), \ldots, d_R(x_1, \ldots, x_n, y)), \tag{14.4}$$

where $f_\vee(a, b, c)$ stands for $f_\vee(f_\vee(a, b), c)$, and, in general, $f_\vee(a_1, \ldots, a_n, a_{n+1}) = f_\vee(f_\vee(a_1, \ldots, a_n), a_{n+1})$.

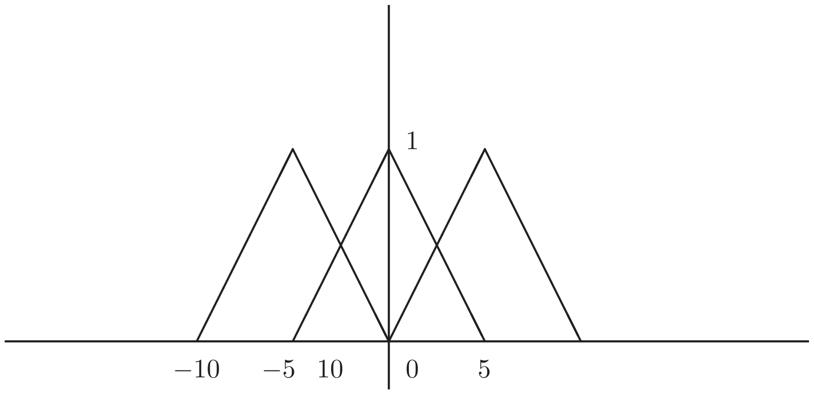
Toy example. In particular, in our toy example,

$$\mu_C(x, u) = f_\vee(d_1(x, u), d_2(x, u), d_3(x, u)),$$

i.e.,

$$\mu_C(x, u) = f_\vee(f_\&(\mu_N(x), \mu_N(u)), f_\&(\mu_{SP}(x), \mu_{SN}(u)), f_\&(\mu_{SN}(x), \mu_{SP}(u))).$$

Numerical example. Let us assume that all three membership functions are piece-wise linear, and that they are described by the following graph:



What is the degree of confidence $\mu_C(4, -2)$ that for $x = 4°$ the control $u = -2°$ is reasonable? According to our formulas, let us first compute the values of the membership functions. From the above general formula for linear extrapolation, we can find the analytical formulas for these membership functions:

- The term "negligible" is described by the following formulas:
  - $\mu_N(x) = 1 + x/5$ for $-5 \le x \le 0$;
  - $\mu_N(x) = 1 - x/5$ for $0 \le x \le 5$;
  - $\mu_N(x) = 0$ for all other x.

- The term "small positive" is described by the following formulas:
  - $\mu_{SP}(x) = x/5$ for $0 \leq x \leq 5$;
  - $\mu_{SP}(x) = 2 - x/5$ for $5 \leq x \leq 10$;
  - $\mu_{SP}(x) = 0$ for all other x.
- The term "small negative" is described by the following formulas:
  - $\mu_{SN}(x) = 2 + x/5$ for $-10 \leq x \leq -5$;
  - $\mu_{SN}(x) = -x/5$ for $-5 \leq x \leq 0$;
  - $\mu_{SN}(x) = 0$ for all other x.

If we use $f_{\&}(a,b) = \min(a,b)$ and $f_{\vee}(a,b) = \max(a,b)$, then we get $\mu_C(4,-2) = \max(d_1, d_2, d_3)$, where

$$d_1 = \min(\mu_N(4), \mu_N(-2)); \quad d_2 = \min(\mu_{SP}(4), \mu_{SN}(-2));$$

$$d_3 = \min(\mu_{SN}(4), \mu_{SP}(-2)).$$

Here, $\mu_N(4) = 0.2$, $\mu_N(-2) = 0.6$, $\mu_S P(4) = 0.8$, $\mu_{SN}(-2) = 0.4$, and $\mu_{SN}(4) = \mu_{SP}(-2) = 0$. Hence,

$$d_1 = \min(0.2, 0.6) = 0.2; \quad d_2 = \min(0.8, 0.4) = 0.4; \quad d_3 = \min(0, 0) = 0,$$

and

$$\mu_C(4,-2) = \max(0.2, 0.4, 0) = 0.4.$$

Exercise

14.2 Compute $\mu_C(3,-3)$ assuming that the properties "negligible", "small positive", and "small negative" are described by the same membership functions, and that we use algebraic product and algebraic sum as AND and OR operations.

14.4. Decision making (defuzzification)

The problem. As a result of applying the previous steps, we get a *fuzzy set* that describes, for each possible control value u, how reasonable it is to use this particular value. In other words, for every possible control value u, we get a degree of confidence $\mu(u)$ that describes to what extent this value u is reasonable to use. In automatic control applications, we want to transform this *fuzzy* information into a *single* value $\bar{u}$ of the control that will actually be applied.

This transformation from a *fuzzy* set to a (*non-fuzzy*) number is called a *defuzzification*. What defuzzification should we apply?

Main idea. In order to find out what defuzzification is the best, let us recall the meaning of the values $\mu(u)$. We have obtained these values *indirectly*, by processing the expert rules formulated for the general inputs. However, we could, in principle, have obtained them *directly*, by:

- asking experts about the possible controls for this very situation, and then
- by applying known elicitation techniques to get the degrees of confidence $\mu(u)$.

In Section 13.3, we described two major elicitation methods: estimating on a *scale* and *polling* (we also described a third, less frequently used, method based on betting).

- Estimation on a *scale* method does not explain why this or that number is chosen by an expert, so if we assume that $\mu(u)$ is obtained by this method, this does not help much in figuring out how we can defuzzify this information.
- The second, *polling* method, as we will see, turns out to be much more helpful.

According to this second elicitation method, to get a value $\mu(u)$, we poll several (N) experts and then define $\mu(u)$ as the ratio $M(u)/N$, where $M(u)$ is the total number of experts who believe that for this particular situation, u is a reasonable control value.

The function $\mu(u)$ is usually different from 0 for *infinitely many* different values u. However, in reality, we can only ask experts about *finitely many* different values. So, to use this interpretation, let us assume that there are only finitely many possible control values.

In this case, as a result of the elicitation, we have several (finitely many) control values u that different experts deemed reasonable. We can order these values into a sequence $u_1, \ldots, u_n$. In this sequence, each value u is proposed by $M(u)$ experts and is, therefore, repeated $M(u)$ times.

If we select a value $\bar{u}$, and the actual best control is u, then we get a control error $e = \bar{u} - u$. Thus, if it turns out that u_i is the best control, we get an error $e_i = \bar{u} - u_i$. We do not know which of the control values is the best, therefore, we would like all these errors e_i to be as small as possible. We cannot get all of these errors e_i equal to 0, so we would like to combine the errors $e_1, \ldots, e_n$ into a reasonable combination $J(e_1, \ldots, e_n)$, and then select $\bar{u}$ for which this combination takes the smallest possible value.

Implementation of the main idea: case of finitely many possible control values. We have already encountered a similar problem in Lesson 3, and we provided arguments in favor of the combination function $J(e_1, \ldots, e_n) = (e_1)^2 + \ldots + (e_n)^2$. This method is called the *least squares* method. If we apply the least squares method to our problem, we will choose the value $\bar{u}$ from the condition that $(\bar{u} - u_1)^2 + \ldots + (\bar{u} - u_n)^2 \to \min$.

This formula can be simplified if we group together the terms $(\bar{u} - u_i)^2$ that correspond to equal values u_i. Since each value u is repeated $M(u)$ times, the minimized sum can be re-written as $J = \sum M(u) \cdot (\bar{u} - u)^2$, where the sum is taken over all possible control values u, and the minimization problem takes the form

$$J = \sum_u M(u) \cdot (\bar{u} - u)^2 \to \min_{\bar{u}}.$$

We cannot directly apply this formula, because we do not know the values $M(u)$. Instead, we only know the values $\mu(u) = M(u)/N$. To use this knowledge, we can divide the minimized quantity J by N.

> We can do that, since an arbitrary function J attains its minimum if and only if J/N attains its minimum, where N is an arbitrary constant.

The problem $J/N \to \min$ can be, thus, re-written as

$$\sum_u \mu(u) \cdot (\bar{u} - u)^2 \to \min_{\bar{u}}.$$

Now, we have formulated the problem of finding the defuzzification $\bar{u}$ as a precise mathematical problem, and we can solve this problem.

One of the main reasons for choosing the least squares method in Lesson 3 was that for this method, equating the derivative with 0 (a standard techniques for solving optimization problems) leads to linear equations. To utilize this advantage, let us differentiate the left-hand side of the minimized quantity with respect to the unknown $\bar{u}$ and equate the result to 0. As a result, we get the equation

$$\sum_u 2 \cdot (\bar{u} - u) = 0.$$

To simplify this equation, we can divide both sides by 2, and then move all the terms that do not contain the unknown $\mu(u)$ into the right-hand side. As a result, we get the equation

$$\bar{u} \cdot \big(\sum_u \mu(u)\big) = \sum_u u \cdot \mu(u),$$

and

$$\bar{u} = \frac{\sum u \cdot \mu(u)}{\sum \mu(u)}. \tag{14.5}$$

This formula is called *centroid defuzzification*, because it resembles a formula from mechanics that describes the *center of mass* $\vec{r}$ of a system of several points with masses m_i at locations $\vec{r}_i$ as

$$\vec{r} = \frac{\sum m_i \cdot \vec{r}_i}{\sum m_i}.$$

Here,

- u is the analog of a location, and
- $\mu(u)$ is the analog of the mass.

Defuzzification for the realistic case of infinitely many possible values of possible control. To get the formula (14.5), we made a simplifying assumption that only *finitely many* different control values u are reasonable (i.e., that only for finitely many values u, we have a positive degree of confidence $\mu(u)$ that u is reasonable). In reality, there are *infinitely many* possible values u. To get a formula for this realistic case, we must take more and more points and then tend this number of points to infinity.

When we take denser and denser values of u, with a step $\Delta u \to 0$, then the sum $\sum \mu(u) \cdot \Delta u$ tends into an *integral* of $\int \mu(u)du$ of the function $\mu(u)$. Thus, for small Δu, the sum $\sum \mu(u)$ (the denominator of the expression (14.5)) is approximately equal to $(\int \mu(u)du)/\Delta u$. Similarly, the sum $\sum u \cdot \mu(u)$ (the numerator of the expression (14.5)) is approximately equal to the ratio

$$(\int u \cdot \mu(u)du)/\Delta u.$$

Thus, when $\Delta u \to 0$, the fraction (14.5) is approximately equal to

$$\bar{u} \approx \frac{(\int u \cdot \mu(u)du)/\Delta u}{(\int \mu(u)du)/\Delta u}.$$

The fraction will not change if we multiply both the numerator and the denominator by Δu. Thus, we get a simplified expression

$$\bar{u} \approx \frac{\int u \cdot \mu(u)du}{\int \mu(u)du}.$$

The smaller Δu, the closer $\bar{u}$ to the right-hand side. Thus, in the limit $\Delta u \to 0$, $\bar{u}$ will be exactly equal to the right-hand side. Hence, we can select

$$\bar{u} = \frac{\int u \cdot \mu(u)du}{\int \mu(u)du}. \tag{14.6}$$

This formula is also called *centroid defuzzification*. It is actually successfully used in fuzzy control.

Warning: centroid defuzzification does not always work. In most real-life situations, centroid defuzzification leads to a meaningful control, but in some cases, it may lead to an unreasonable control.

Let us give a simple example. Suppose that we are designing an automatic controller for a car. If the car is traveling on an empty wide road, and there is an obstacle straight ahead (e.g., a box that fell from a truck), then a reasonable idea is to *swerve* to avoid this obstacle. Since the road is empty, there are two possibilities:

- we can swerve to the right; and
- we can swerve to the left.

For swerving, the control variable u is the angle to which we steer the wheel. Based on the distance to the obstacle and on the speed of the car, an experienced driver can describe a reasonable amount of steering u_0.

In reality, u_0 will probably be a fuzzy value, but for simplicity, we can assume that u_0 is precisely known.

Thus, as a result of formalizing expert knowledge, we conclude that there are two possible control values:

- the value u_0 with degree of confidence $\mu(u_0) = 1$, and
- the value $-u_0$, with degree of confidence $\mu(-u_0) = 1$.

If we apply the defuzzification formula (14.5) to this situation, we get

$$\bar{u} = \frac{u_0 \cdot \mu(u_0) + (-u_0) \cdot \mu(-u_0)}{\mu(u_0) + \mu(-u_0)} = \frac{u_0 \cdot 1 + (-u_0) \cdot 1}{1 + 1} = 0.$$

So, the recommended control means that *no swerving* will be applied at all, and the car will run straight into the box.

To avoid such situations, we must *modify* the centroid defuzzification. We could have screened out the value $\bar{u} = 0$ because for this value, $\mu(\bar{u}) = 0$. Thus, instead of using a simply centroid defuzzification, we can do the following:

- First, we apply defuzzification to the original membership function $\mu(u)$.
- Then, we check whether the resulting control value $\bar{u}$ is reasonable, i.e., whether the degree of confidence $\mu(\bar{u})$ is big enough (e.g., larger than some pre-defined value μ_0).
  - If $\mu(\bar{u}) \geq \mu_0$, then we apply the control $\bar{u}$.
  - If $\mu(\bar{u}) < \mu_0$, this means that there are several areas of reasonable control separated by a gap, and $\bar{u}$ happens to be in this gap. In this case, instead of applying the the centroid defuzzification to the entire *membership* function $\mu(u)$, we do the following:
    - we select *one of the areas*, and then
    - we apply centroid defuzzification only to value u from this area.

This idea was first proposed and successfully implemented by John Yen.

14.5. Summary: fuzzy control

Let us summarize what we have learned in these two lessons:

- We start with the if-then expert rules of the type "If x is small, and ..., then y is small". In general, each of these rules can be represented by a formula
$$A_{r1}(x_1)\&\ldots\&A_{rn}(x_n) \to B_r(y), \tag{13.4}$$
where $x_1, \ldots, x_n$ are inputs, and A_{ri} and B_r are words that describe properties of inputs and output.

- For each words w used in these rules, we pick several values $x^{(1)}, \ldots, x^{(k)}$, and use one of the elicitation techniques described in Lesson 13 to determine the degrees of confidence $\mu_w(x^{(1)}), \ldots, \mu_w(x^{(n)})$ that these values satisfy the property w.

- Then, we use some extrapolation technique to determine the membership functions $\mu_w(x)$ (that describe the degrees of confidence that different values of x satisfy the property w).

- We choose AND and OR operations $f_\&(a,b)$ and $f_\vee(a,b)$.

- For each rule r, and for each possible values of input and output, we compute the *firing degrees* $d_r(x_1, \ldots, x_n, y)$ using the formula
$$d_r(x_1, \ldots, x_n, y) = f_\&(\mu_{r1}(x_1), \ldots, \mu_{rn}(x_n), \mu_r(y)), \tag{14.3}$$
and then we compute the membership function for control as
$$\mu_C(x_1, \ldots, x_n, y) = f_\vee(d_1(x_1, \ldots, x_n, y), \ldots, d_R(x_1, \ldots, x_n, y)). \tag{14.4}$$

- For every input $x_1, \ldots, x_n$, we get a function $\mu_C(y)$ that describes the degree of confidence that this very y is a reasonable control. To get a single recommended control value $\bar{y}$, we use a formula
$$\bar{y} = \frac{\int y \cdot \mu_C(y)dy}{\int \mu_C(y)dy} \tag{14.6a}$$
or a more complicated defuzzification method described in the previous section.

Comment. It is quite possible that the resulting control is sometimes inadequate. There are two reasons for this:

- First, when an expert formulates the rules, he usually remembers *specific* rules but sometimes forgets to explicitly mention *common sense* rules that are absolutely evident to any human, but that need to be explicitly spelled out for the computer.
- Second, all knowledge elicitation methods are *approximate*, and if we add distortions caused by this approximate character on each step of fuzzy control methodology, we may end up with a rather distorted representation of expert's control.

In view of this possibility, before implementing this control, we must first *test* it. If it turns out that in some situations, the resulting control is inadequate, we have two options:

- If the inadequacy is *huge*, this probably means that we are missing or misinterpreting some of the rules. In this case, we need to confront the experts with these results. Since the rules that the experts have formulated lead to not adequate results, the experts will be able to either *modify* these rules, or *add* new rules that cover these situations.
- If the inadequacy is *small*, then probably, the experts' rules were adequate, and the inadequacy is caused by the approximate character of the expert system methodology. In this case, to make a control better, we can *tune* the parameters of the resulting control.

For further reading on the fundamentals of fuzzy control, see:

- D. Driankov, H. Hellendoorn, and M. Reinfrank, *An introduction to fuzzy control* (Springer-Verlag, Berlin, 1993).
- R. Palm, D. Driankov, and H. Hellendoorn, *Model based fuzzy control* (Springer-Verlag, Berlin, Heidelberg, 1997).

Exercise

14.3 For the toy thermostat example, write a program that will compute the optimal control value $\bar{u}$ for every given x. To compute the integrals, use either an analytical expression or some simple numerical method.

14.4 Apply fuzzy control methodology to design an extrapolation algorithm. Namely, we start with the patterns $(x^{(i)}, y^{(i)})$, and for each pattern, we formulate the rule "if x is close to $x^{(i)}$, then y is close to $y^{(i)}$". To describe a notion "x is close to x'", use a simple membership function $\mu(\Delta x)$, where $\Delta x = x - x'$ (e.g., a triangular function $\mu(\Delta x)$).