Building fuzzy systems and controllers

A **fuzzy system** is a system that makes use of fuzzy logic. In this chapter, we're going to examine how to build such a system. We also look at the working principles of fuzzy controllers.

1 Construction of fuzzy systems

1.1 Basic fuzzy system construction

When building a fuzzy system, the first thing that needs to be defined is the **structure** of the system. This structure consists of the following parts.

- The number and type of **input and output variables**.
- The **structure of the rules**. That is, the kind of fuzzy model that is used.
- The number of **linguistic variables** and the number and type of **membership functions** for each variable. (E.g. whether triangular or trapezoidal membership functions are used.)
- The type of **inference mechanism** that is used.
- The **defuzzification method**.

Once the structure has been set up, the available knowledge should be formulated as a set of 'if-then' rules. This then results in a fuzzy system. But it's not done yet. Based on available test data, the parameters of the system can be fine-tuned. (With parameters, we mainly mean the parameters of the membership functions, like the position of the top at the triangular membership function and such. But in some models, the if-then rules also have parameters.) Finally, the system needs to be evaluated. If it does not meet the expectations, then a new system should be created with a somewhat different structure.

1.2 The least-squares estimation of consequents

Let's suppose that we are using an affine Takani-Sugeno model. So we have a set of K rules with consequents of the form $y = \mathbf{a_i}^T\mathbf{x} + \mathbf{b_i}$. We need to set these parameters $\mathbf{a_i}$ and b_i. This can be done by using a set of N input-output data pairs $(\mathbf{x_i}, y_i)$. We can put these pairs into an input matrix $\mathbf{X} = [\mathbf{x_1}, \ldots, \mathbf{x_N}]^T$ and an output vector $\mathbf{y} = [y_1, \ldots, y_N]^T$. Now, because the models are linear, we can use the least-squares method to find optimal values for $\mathbf{a_i}$ and b_i.

First, let's define the matrix $\mathbf{\Gamma_i}$ as the $N \times N$ diagonal matrix having the normalized membership degrees $\gamma_i(\mathbf{x_j})$ on its diagonal. We also append the matrix $\mathbf{X}$ with a column of ones to get $\mathbf{X_e} = [\mathbf{X}\ \mathbf{1}]$. Now, define the matrix $\mathbf{X'}$ as

$$\mathbf{X'} = \begin{bmatrix} \mathbf{\Gamma_1 X_e} & \mathbf{\Gamma_2 X_e} & \ldots & \mathbf{\Gamma_K X_e} \end{bmatrix}. \tag{1.1}$$

The consequent parameters, which need to be found, are put into one big vector, being

$$\theta = \begin{bmatrix} \mathbf{a_1}^T & b_1 & \mathbf{a_2}^T & b_2 & \ldots & \mathbf{a_K}^T & b_K \end{bmatrix}. \tag{1.2}$$

The least-squares equation which we want to solve is $\mathbf{y} \approx \mathbf{X'}\theta$. The solution for θ is thus given by

$$\theta = \left((\mathbf{X'})^T\mathbf{X'}\right)^{-1}(\mathbf{X'})^T\mathbf{y}. \tag{1.3}$$

This trick can also be used for the singleton model. But now we simply omit $\mathbf{a_i}$ and set $\mathbf{X_e} = \mathbf{1}$.

1.3 Defining linguistic terms and membership functions

It is often difficult to choose which linguistic terms to use and which membership functions to give them. In **template-based modeling**, we use a simple technique. We simply define a set of K linguistic terms A_i. The membership functions of A_i are now distributed such that the whole interval of possible inputs x is covered. Every membership function A_i then gets its own if-then rule.

The question remains how to distribute the membership functions. If no knowledge on the system is present, the functions are distributed evenly over the interval. The big downside of the template-based modeling now is that the number K of linguistic terms may grow very fast.

Another way to define rules is by using **fuzzy clustering**. We simply take N samples of data and divide them over K fuzzy sets A_i using a fuzzy clustering technique. Now we can define K rules; one for each fuzzy cluster A_i. When also the Takagi-Sugeno model is applied, we will get rules like

$$\textbf{if } x \text{ is } A_i \textbf{ then } y = a_i x + b_i. \tag{1.4}$$

1.4 Applying fuzzy system construction to systems

Let's suppose that we have some system in which the output $y(t+1)$ can be modeled as

$$y(t+1) = f(y(t), y(t-1), u(t), u(t-1)). \tag{1.5}$$

Such a model is called a **second-order NARX model**. In the above equation, $y(t+1)$ is called the **regressand**, while $y(t)$, $y(t-1)$, $u(t)$ and $u(t-1)$ are the **regressors**. We can define the **regressor matrix X** and the **regressand vector y** up to time N_d as

$$\mathbf{X} = \begin{bmatrix} y(2) & y(1) & u(2) & u(1) \\ y(3) & y(2) & u(3) & u(2) \\ \vdots & \vdots & \vdots & \vdots \\ y(N_d-1) & y(N_d-2) & u(N_d-1) & u(N_d-2) \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y(3) \\ y(4) \\ \vdots \\ y(N_d) \end{bmatrix}. \tag{1.6}$$

Now we can construct our fuzzy system in one of the normal ways. So we see that fuzzy systems can also be used to approximate system dynamics.

Sometimes we can help the fuzzy system a bit. Let's suppose that we have some system $y = f(x)$. We also know, thanks to physical insights, that the output y is roughly proportional to x^2. In this case, it might be better to use x^2 as input instead of x. (Fuzzy systems are better at approximating linear functions than they are at approximating quadratic functions.) So, by simply changing the input of the system, the performance will most likely increase. This trick of using physical insights to give the system an easier job is called **semi-mechanistic modeling**.

2 Construction of fuzzy controllers

2.1 Why use fuzzy controllers?

In convential control theory, we use mathematical models to design a controller. However, if it is hard to obtain a model, or if the system is highly nonlinear, this approach doesn't work so well. Luckily, fuzzy models can be used to approximate nonlinear functions. So, we can make a fuzzy controller! A **fuzzy controller** is a controller that contains a mapping that has been defined using if-then rules.

In the world of control systems, a lot of systems are nonlinear. So, the demand for nonlinear control methods is big. Such methods need to have several properties. First of all, it would be nice if they don't

need too many input variables to obtain good results. Also, the method needs to be able to deal with nonlinearities well, it should have a good learning/training rate, it should not be too computationally intensive, and several more criteria need to be met. Fuzzy logic is a method that performs quite well on these criteria.

2.2 The Mamdani controller

Several types of fuzzy controllers exist. One important example is the **Mamdani controller**. It is usually used as a feedback controller.

As input, the Mamdani controller generally receives the error signal e. This error signal is then processed by **dynamic pre-filters**. These pre-filters often **scale** the data, for example to put it on the normalized interval $[-1, 1]$. Also, **dynamic filtering** is often applied, where quantities like $\dot{e}$ and $\int e$ are derived. Other pre-filtering methods can be applied as well. The resulting parameters are then fed into the part of the fuzzy controller known as the **static map**. The output of the static map is then fed to **dynamic post-filters**. These filters can again scale the data and/or apply dynamic filtering.

Let's take a closer look at the static map. It is important to define it properly. So, we're going to look at the individual steps needed to design the static map.

- First, the necessary **inputs and outputs** need to be selected. For conventional linear controllers, it can be advantageous to use integral or derivative gains. Similarly, for fuzzy logic, it can sometimes be advantageous to also use parameters like $\dot{e}$, $\int e$ and/or others as inputs. However, things can get complicated if too many input variables are used. If this is the case, then it may be worthwhile to split up the system into several subsystems and let the output from one subsystem be the input to the other.
- Second, the **number of linguistic terms** for every input variable needs to be set. If too few terms are used, then the fuzzy system doesn't have a lot of flexibility – it can't approximate all kinds of functions. However, if too many linguistic terms are used, the rule base will become rather big.
- For every linguistic term, a **membership function** needs to be selected. For computational reasons, triangular and trapezoidal functions are usually preferred to bell-shaped functions.
- A very important step is to design the **rule-base**. This is generally done based on the knowledge of an expert. Also, a model of the system may be used.
- Finally, the fuzzy controller needs to be **tuned**. This step is just as important as the tuning of the gains in a conventional PID controller. The tuning of a fuzzy controller can be difficult because sometimes, by changing only one variable, the whole system changes. But luckily, the effects of changing parameters in a fuzzy controller are usually quite localized.

2.3 The Takagi-Sugeno controller and supervisory control

The Takagi-Sugeno fuzzy controller is somewhat similar to gain scheduling. Rules now take the form

$$\textbf{if } e \text{ is } \mathit{Low} \textbf{ then } u = a_i e + b_i \tag{2.1}$$

or something similar. Each rule is valid for only a small region of the controller's input space. So, every region of the controller's input space more or less has its own control law. The general controller then simply interpolates between these control laws.

Something entirely different is the **supervisory controller**: it is a secondary controller. It augments an existing (conventional) controller. To do this, the supervisory controller usually defines certain parameters of the existing controller. For example, it may define the proportional and derivative gains K_p and K_d of a conventional PID controller, based on the current state of the system. Rules can thus take the form of

$$\textbf{if } \text{process output is } \mathit{High} \textbf{ then } \text{reduce } K_p \ \mathit{Slightly} \textbf{ and } \text{increase } K_d \ \mathit{Moderately}. \tag{2.2}$$