Computation of Centroids for Interval Type-2 Fuzzy Membership Functions

By

Mayurika Bhar

Examination Roll No. : **M6IAR19013**

Registration No. : **137302 of 2016-2017**

Under the Guidance of

Prof. Amit Konar

Thesis submitted in partial fulfillment of the requirement for the award of the Degree of Master of Technology in Intelligent Automation and Robotics under Electronics and Telecommunication Engineering

Department of Electronics and Telecommunication Engineering
Jadavpur University
Kolkata – 700032
May, 2019

Faculty of Engineering and Technology
Jadavpur University
Kolkata – 700032

Date :

I hereby recommend that the thesis under our supervision by Mayurika Bhar, entitled **"Computation of Centroids for Interval Type-2 Fuzzy Membership Functions"**, be accepted in partial fulfillment of the requirement of the degree of Master in Intelligent Automation and Robotics under the department of Electronics and Telecommunication Engineering.

Prof. Amit Konar
Supervisor
Dept. of Electronics and Telecommunication Engineering, Jadavpur University

----
Prof. Amit Konar
Coordinator, IAR course
Dept. of Electronics and Telecommunication Engineering, Jadavpur University

----
Dr. Sheli Sinha Chaudhuri
Head of the Department
Electronics and Telecommunication Engineering, Jadavpur University

----
Prof. Chiranjib Bhattacharjee
Dean, Faculty Council of Engineering and Technology, Jadavpur University

Faculty of Engineering and Technology
Jadavpur University
Kolkata – 700032

CERTIFICATE OF APPROVAL

The foregoing thesis is hereby approved as a credible study of engineering subject to warrant its acceptance as a pre-requisite to obtain the degree for which it has been submitted. It is understood by this approval the undersigned do not endorse or approve any statement made, opinion expressed or conclusion drawn therein , but approve the thesis only for the purpose for which it is submitted.

Signature of the Examiners

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC THESIS

I hereby declare that this thesis titled "**Computation of Centroids for Interval Type-2 Fuzzy Membership Functions**" contains literature survey and original research work by the undersigned candidate, as part of his Degree of Master of Technology in Intelligent Automation and Robotics.

All information have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Mayurika Bhar
Examination Roll No: M6IAR19013

Thesis Title: COMPUTATION OF CENTROIDS FOR INTERVAL TYPE-2 FUZZY MEMBERSHIP FUNCTIONS

Date:
Place: Kolkata

Signature of the candidate

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement made the work a success. I would like to express deep gratitude and respect to all those people who guided, inspired and helped me for the completion of my project work.

It is a great pleasure to express my deepest gratitude to my project guide **Prof. Amit Konar**, Department of Electronics and Telecommunication Engineering, for giving me the opportunity to work on this project. I would like to acknowledge my sincere appreciation for his valuable advice, phenomenal guidance, and constructive suggestions throughout the course of the thesis work. I was at greatest liberty to exercise thoughtful and scientific approach to the problem.

I express my sincere thanks to **Dr. Sheli Sinha Chaudhuri,** Head of the Department of ETCE for providing me the necessary facilities in the department. I am also thankful to all the staff members of the Department of ETCE and to all my well-wishers for their inspiration and help. I would like to thank my lab mates for providing me moral support towards the fulfillment of this project work.

Finally, and most importantly, I wish to express my deep gratitude to my parents, who always stood by me throughout my life and guided me in my time of crisis giving me inspiration and encouragement in writing this thesis.

Date:
Place: Kolkata

Mayurika Bhar
M.Tech. in Intelligent Automation and Robotics
(E.T.C.E)
Examination Roll No: M6IAR19013
Registration No: 137302 of 2016-2017

Contents

Chapter 2: Type-2 Centroids

Chapter 3: Matlab Simulation and Comparison

Chapter 4: Conclusions and Future Scopes

Synopsis of the Thesis

In research field the type reduction of defuzzification process in Interval Type-2 Fuzzy Sets (IT2 FSs) is a challenging domain and has grabbed attention. Karnik-Mendel (KM) algorithm is the basic approach for Centroid type reduction. There are several methods for type reduction (T2 FS to T1 FS) in IT2 FSs, such as Centroid type reduction, Height type reduction, Center-of-sets type reduction etc.

All the algorithms mentioned in this thesis are primarily based on KM algorithm for Centroid type reduction. The algorithms are broadly classified into three categories. In first category five enhanced to KM algorithms are mentioned. Through a comprehensive overview and a comparison described here, it is examined that they all reduce the computational cost over KM algorithm in real world of uncertainties. Moreover the computational times for all these methods are minimized. Among these enhanced methods the EODS method is the fastest methods which require less computational cost. EIASC algorithm is much simpler than EODS algorithm. But it is 1.2 times slower than EODS algorithm. EIASC may also be preferred for understanding and implementation for centroid type reduction in defuzzification process and certain cost reduction applications.

In the second category, eleven alternative type reduction (TR) methods are examined. Among all the alternative TR algorithm WT and NT methods are considered as the fastest method as they have already minimized the overall computational cost. It is also observed that WT and NT methods are faster algorithms than enhanced to KM algorithm, EODS method. Additionally, properties such as stability and robustness of BMM method are studied here for which it can also be preferred for cost reduction applications.

The third category consists of a simplified structure for IT2 FLCs, which can be merged with any algorithm in the first or second category. A simplified IT2 FLC can save quite a large amount of computational cost over a full IT2 FLC, especially when the number of rules is high. It is tested that simplified WT-NT method has the fastest speed over all other algorithms.

The thesis contains four chapters. In Chapter 1, a general overview of fuzzy logic and fuzzy sets are described. The type-1 (T1) fuzzy logic sets (FLSs) and the T1 fuzzy logic systems (FLCs) are explained in this chapter. The membership functions (MFs) of T1 FSs, which are a crisp logic are also presented in this section. The type-2 (T2) FSs, interval type-2 fuzzy sets (IT2 FSs) and interval type-2 fuzzy logic systems (IT2 FLSs) are enormously explained in this section. A brief introduction of Cardinality, Variance, Fuzziness and Skewness of Interval Type-2 Fuzzy Set has been outlined at the end of this chapter. In Chapter 2 all the three categories of Centroid type reduction algorithms are briefly presented. In Chapter 3, a comprehensive overview and comparison of three categories of methods to reduce their computational cost are illustrated. All the comparisons are done here in both control surface and evolutionary IT2 FLCs using both Gaussian MFs and Trapezoidal MFs. In Chapter 4, the conclusion and the real world applications of IT2 FSs are enlightened. Lastly the future scope of this project is directed in this chapter. All the Matlab Codes and outputs of centroid using different type reduction algorithms are provided in APPENDIX A. Comparisons of different type reduction algorithms and KM algorithms using Gaussian MFS and Trapezoidal MFs are graphically shown in APPENDIX B.

List of Tables

List of Figures

Chapter 1

Introduction to Type-2 Fuzzy Sets

The chapter provides an introduction of fuzzy set, fuzzy membership function and different types of membership function. It also describes Type-2 Fuzzy Set, Interval Type-2 Fuzzy Set, Upper and Lower membership function of Interval Type-2 Fuzzy Set. Footprint of Uncertainty is described which measures the real world uncertainty. Fuzzy Logic System of both Type-1 and Interval Type-2 fuzzy and suitable comparisons are mentioned in this chapter. A Wavy-Slice representation of Interval Type-2 Fuzzy Set is pointed here. A brief introduction of Cardinality, Variance, Fuzziness and Skewness of Interval Type-2 Fuzzy Set has been outlined at the end of the chapter.

1.1 Introduction

Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1 inclusive. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. The term fuzzy logic was introduced with the 1965 proposal of fuzzy set theory by Lotfi Zadeh. It is based on the observation that people make decisions based on imprecise and non-numerical information, fuzzy models or sets are mathematical means of representing vagueness and imprecise information, hence the term fuzzy. These models have the capability of recognizing, representing, manipulating, interpreting, and utilizing data and information that are vague and lack certainty.

The traditional Type-1 Fuzzy Logic Controllers (FLCs), which use crisp type-1 fuzzy sets, whose membership function is crisp logic, cannot handle high levels of uncertainties appropriately. However it has been shown that a Type-2 FLC using Type-2 fuzzy sets whose membership function is Type-1 fuzzy sets can handle such uncertainties better and thus fabricate a better performance. Most of the real world applications for uncertainties have been done on the basic idea of Type-2 fuzzy sets. As such, Type-2 FLCs are considered to have the potential to overcome the limitations of Type-1 FLCs and produce a new generation of fuzzy controllers with improved performance for many applications which require handling high levels of uncertainty. Interval Type-2 fuzzy set is the special type of Type-2 fuzzy set whose membership functions is an interval. Interval Type-2 fuzzy sets are widely used for high level of real world uncertainties, really in Computing With Word (CWW).

1.2 Conventional Sets

A set is defined as a collection of objects which have one or more common characteristics. The objects that belong to the particular set are called as members or objects. The characteristics used to define a set should be sufficient to identify its members. For example, the persons enrolled for a particular course may together be called a STUDENT set. This is a set because we can easily identify if a person is a student or not by identifying his/her name in the registered book. But the SMALL RIVERS of a country is not a set as a river of the country does not belongs to the particular set until we define the length of the river to be considered as SMALL.

Let S be a set and x be a member of S, we can define this relation by following mathematical operation

$$x \in S \tag{1.1}$$

Also let y be a member of set T, such that for all y, y is also a member of S. Then T is called the subset of S, denoted by

$$T \subseteq S \tag{1.2}$$

If T is a subset of S but every element of S are not present in T then T is called the proper subset of S and it is defined by the following mathematical relation

$$T \subset S \tag{1.3}$$

If every element contained in T is an element of S and every element contained in S is an element of T, and then set S and set T are equal, denoted by

$$T = S \tag{1.4}$$

For any two sets S and T if there exist at least one common element x of both S and T, then it is written as

$$x \in (S \cap T) \tag{1.5}$$

where $\cap$ denotes the logical intersection operation.

For any two sets S and T if there exist at least one element x, such that x is a member of S or T, then we can write

$$x \in (S \cup T) \tag{1.6}$$

where $\cup$ denotes the logical union operation.

A universal set U in a domain is a set that includes all possible members of that particular domain. In other words all sets in a given domain are subsets of the universal set U.

1.3 Brief discussion on Fuzzy Sets

A fuzzy set is a collection (i.e., a group) of objects along with a description, called a membership function, of how much each member of the group belongs in the collection.

A fuzzy set extends the binary membership: {0,1} of a conventional set to a spectrum in the interval of [0,1]. Further unlike a conventional set, all the elements of universal set U are members of a given set A. Thus for each element $x \in U$,

$$0 \leq \mu_A \leq 1 \tag{1.7}$$

Definition 1.1 A fuzzy set A is a set of ordered pairs, given by

$$A = \{(x, \mu_A(x)) : x \in X\} \tag{1.8}$$

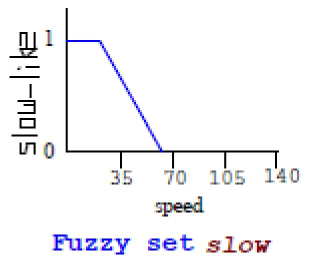
where X is a universal set of objects (also called the universal of discourse) and $\mu_A(x)$ is the grade of membership of the object x in A. Usually, $\mu_A(x)$ lies in the closed interval of [0.1].

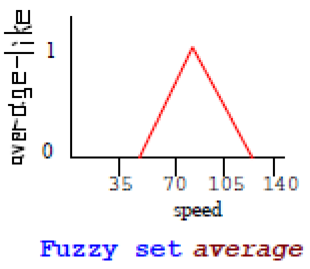
The fuzzy set can be easily described by the following example. First we consider there are three types of cars Indy, Lamborghini and Mercedes that are shown in the figure Fig 1.1.

Figure 1.1 Different cars with different speed

Now it should be identified which car is “fast” or “not fast” or “not slow” or “not very fast” or “very fast” etc. Computers operate on a very different level than people do. Rather than using any sort of fuzzy terms to describe things, a computer makes a very clear separation between groups of objects. In the basic language of computers, every signal is either a zero or a one—there is nothing in-between. By using this crisp logic computer cannot classify the cars to “not fast” or “not slow” or “not very fast” or “very fast” etc. The computer would classify the cars as being either "fast" or "not fast", depending on whether or not the car could exceed a certain speed. If the computer's rule was "If a car can travel faster than 140 mph, then it is fast", and the car Lamborghini could only go 119 mph, the computer would label it as "not fast". We know that car Lamborghini is actually pretty fast, but the computer can make no distinction, and puts the car that can go 119 mph into the same category as a car that can only go 10 mph. In this example the computer separates all cars into one of two groups (this is known as crisp logic, where everything is either true or false), but people put cars into several groups, and sometimes we put one car into more than one group at a time.

Using the example of the cars we can decide that a car wasn't just as fast or not as fast, but that there were different levels in-between these two extremes of how fast a car is. These different levels make up fuzzy sets. For example, Fig. 1.2 shows membership functions for three different fuzzy sets, slow, average, and fast. These sets and their associated membership functions describe the degree of similarity of a car to one that is slow, average, or fast.





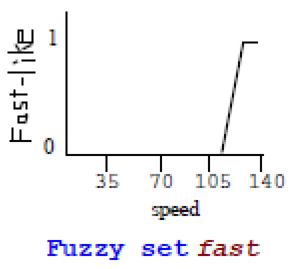
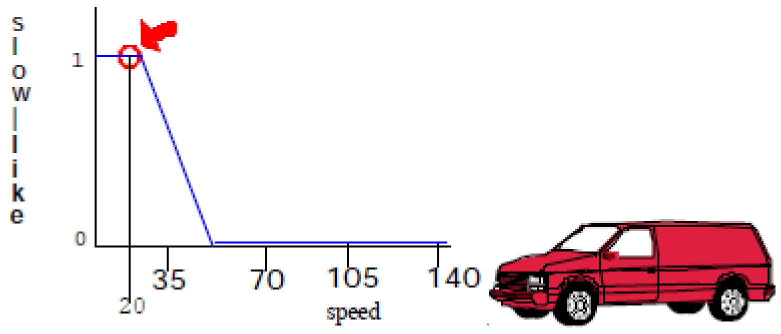


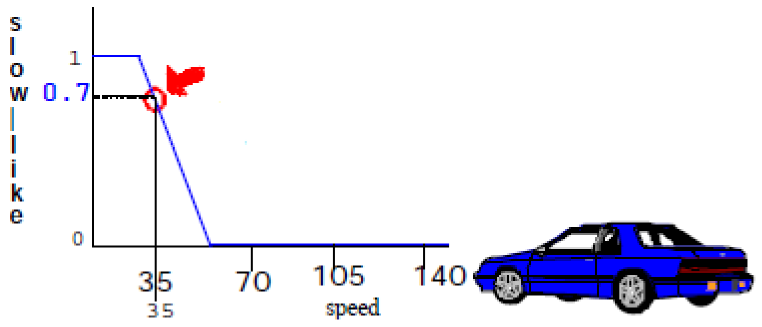
Figure 1.2 Different fuzzy sets for cars, based on the speed of the car. The horizontal axis displays different speeds in miles per hour, and the vertical axis represents membership values ranging from zero to one (unity). The membership function associates the speeds in each set with a membership value. The closer the membership value is to unity the more slow-like, average-like, or fast-like is the identified car; whereas, the closer the membership function is to zero the less slow-like, average-like, or fast-like is the well-known car.

A membership value for any car describes how much the car belongs in that particular set and can vary from a membership value of zero (meaning it does not belong there at all) to a membership value of one, or unity (meaning that it completely belongs in that fuzzy set, and is described perfectly by that set). The membership value for this car has in any of these sets is determined by the car's speed.

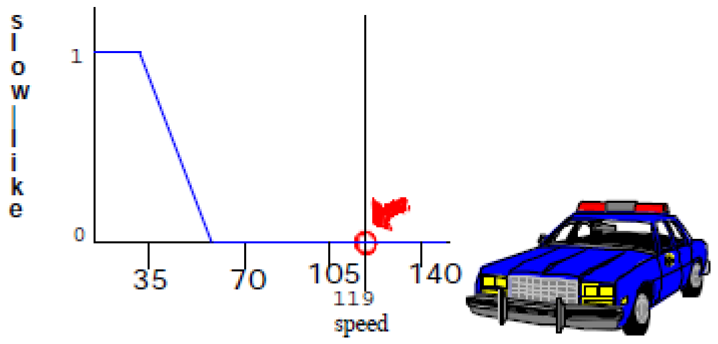
For example, (see Fig. 1.3) a car that has a top speed of 20 mph has a membership value of 1.0 in the set slow. A car with a top speed of 35 mph has a membership value of 0.7 in the set slow, and a car with a top speed of 119 mph has a membership value of 0.0 in the set slow, because it does not belong in this set at all.



Here the car's top speed is 20 mph, and its membership value in the set slow is 1.



Here the car's top speed is 35 mph, and its membership value in the set slow is 0.7.



Here the car's top speed is 119 mph and its membership value in the set slow is 0, because it is not in the set.
Figure 1.3 Fuzzy sets representation of the cars

The more interesting thing about the fuzzy sets is a car can be in more than one fuzzy set at the same time. In traditional "crisp" sets, either an object is in the set, which has a membership value of 1.0 or is out of the set, which has a membership value of 0.0. With crisp sets no object can be kind of in one set and kind of in another. With fuzzy sets, this is no longer true. When the membership functions for the three different fuzzy sets slow, average, and fast are all put on the same axis (which is a measure of speed), as in Fig. 1.4, it can be seen that the sets overlap. If a car's top speed is between 50 mph and 60 mph, the car is described to some extent by both sets slow and average. Similarly, if a car's top speed is between 110 and 125 mph it is in the set average to some degree and in the set fast to some other degree.

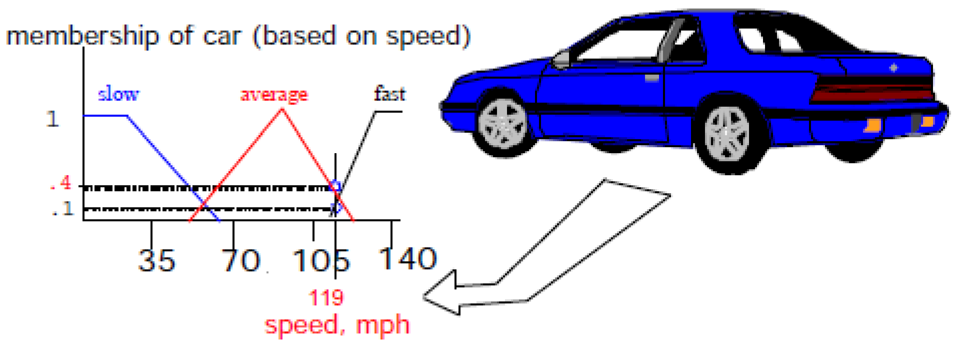


Figure 1.4 Fuzzy sets slow, average, and fast. The car Lamborghini has membership values in the fuzzy sets average and fast.

Here we notice that the car Lamborghini, whose top speed is 119 mph, is described simultaneously by the fuzzy sets average and fast (Fig. 1.4). It can be seen that the car has a membership value of 0.4 in the set average. Because this value is less than 1, the car is not a perfect example of an average-speed car, but it still belongs in the set average. The car also has a membership of 0.1 in the set fast. This means the car also belongs to the group of fast cars, but not very strongly, because most fast cars have much higher top speeds than the Lamborghini has. This car also has a membership value of 0.0 in the fuzzy set slow, because it does not belong in that set at all.

Because the car is in two sets, a choice has to be made about how to finally characterize it. In conversation, we usually don't refer to a car like Lamborghini as "40% average and 10% fast". Instead, we compare in our minds how much the car belongs in the average set and how much it belongs in the fast set. Here we don't use any numbers for this comparison; but, we do have some measure in our minds of how much the car belongs in each particular set. In this case, we decide the car belongs more in the average set than in the fast set. We realize that the car isn't only average or only fast, so we can probably describe it as an "above-average-speed car" (Fig.1.5).

If we compare the two membership values given for the car in this example, the 0.4 membership value in the average set is bigger than the 0.1 membership value in the fast set. If we decide that the car is described more by the set with the bigger membership number, we arrive at the same conclusion just we have reached. Since the car is mostly average and slightly fast, the car Lamborghini is an "above average speed car". Membership values can therefore be used to reach conclusions that are similar to the conclusions we make without ever considering numbers. This procedure can be implemented by computers because they deal with numbers.

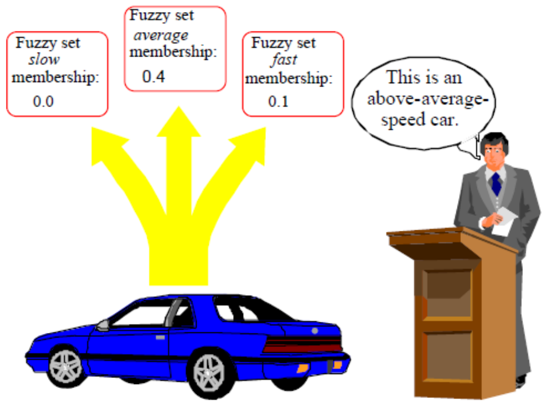

Fuzzy set *slow* membership: 0.0
Fuzzy set *average* membership: 0.4
Fuzzy set *fast* membership: 0.1
This is an above-average-speed car.

Figure 1.5 Different fuzzy membership values for Lamborghini, and how we would describe that car.

The main things about the fuzzy set is described below

1. The membership function of a fuzzy set describes how much an item belongs to the particular set.

2. The membership values range from 0 to 1 and it provide a degree of similarity of an item to the fuzzy set.

3. At the same time an item can belong in more than one fuzzy set.

1.4 Definition of Membership Function

The grade of membership μ_A (x) maps the object or its attribute x to positive real numbers in the interval [0, 1]. It has mapping characteristics like a function and it is called membership function. A formal definition of membership function is given below

Definition 1.2 A membership function $\mu_A(x)$ is described by the following equation:

$$\mu_A(x) : x \rightarrow [0,1],\ \ x \in X \qquad (1.9)$$

where x is a real number which describes an object or its attribute and X is the universe of discourse and A is a subset of X.

There are basically two types of membership function; i) Continuous and ii) Discrete membership function. For example the roll numbers of the students in a class is a discrete universe and the membership function is represented by discrete form. But the height of persons is a continuous universe as height may take up any values between 4' to 8'. The membership function or height is represented by continuous form.

1.5 List of different types of Membership Function & its explanation

The membership function can take any form like Triangular, Gaussian (bell-shaped), S-function, γ-function, L-function, Π-function etc.

1.5.1 The γ- Function

The γ-function can be represented by only two parameters α and β, which is defined by the following equation and graphical representation, given in Fig 1.6.

$$\left.\begin{aligned} \gamma(u;\alpha,\beta) &= 0 && u \le \alpha, \\ &= (u-\alpha)/(\beta-\alpha) && \alpha < u \le \beta, \\ &= 1 && u > \beta \end{aligned}\right\} \qquad (1.10)$$

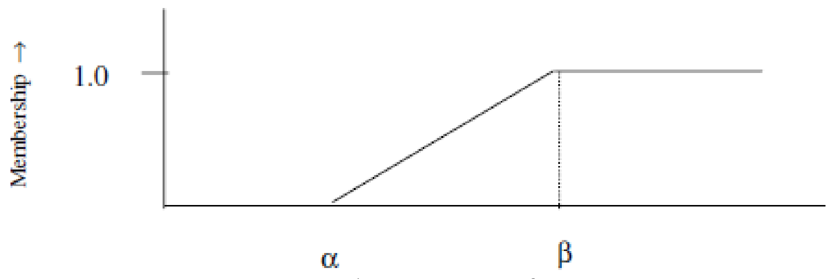


Figure 1.6 The γ-membership function

1.5.2 The S- Function

The S-function is a smooth function of γ-function. It can be written as

$$\left.\begin{aligned} S(u;\alpha,\beta,\gamma) &= 0 && u \le \alpha, \\ &= 2[(u-\alpha)/(\gamma-\alpha)]^2 && \alpha < u \le \beta, \\ &= 1- 2[(u-\alpha)/(\gamma-\alpha)]^2 && \beta < u \le \gamma, \\ &= 1 && u > \gamma \end{aligned}\right\} \qquad (1.11)$$

Generally $\beta = (\alpha+\gamma)/2$ is considered in most application in fuzzy logic. The graphical representation is given below (Fig 1.7).

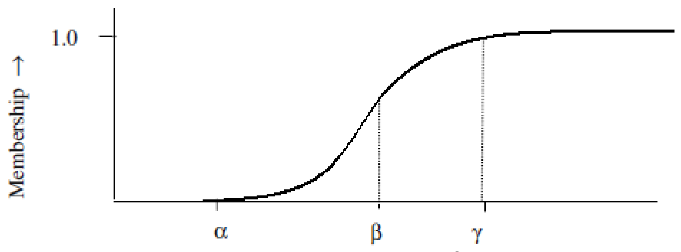


Figure 1.7 The S-membership function

1.5.3 The L- Function

This function is the converse of the typical γ-function. The mathematical (Eq:1.12) and graphical representation (Fig 1.8) is given below

$$L(u; \alpha, \beta) = \left.\begin{aligned} &1 && u < \alpha, \\ &= (\alpha-u)/(\beta-\alpha) && \alpha \le u \le \beta, \\ &= 0 && u > \beta \end{aligned}\right\} \qquad (1.12)$$

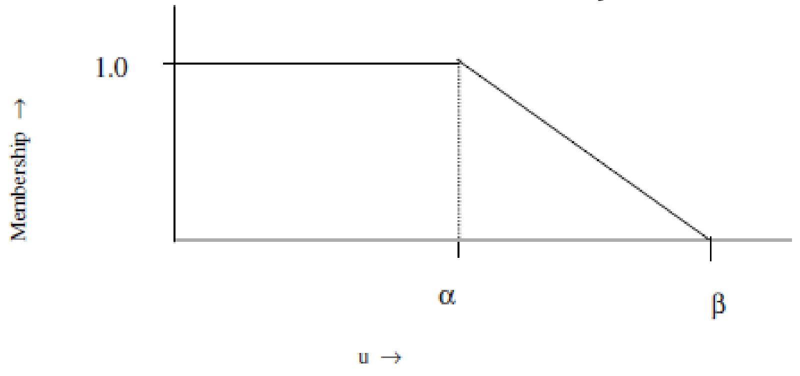


Figure 1.8 The L-membership function

1.5.4 The Triangular Membership Function

The triangular membership function which is also called bell-shaped function with straight lines can be defined as follow

$$\wedge\,(u; \alpha, \beta, \gamma) = \left.\begin{aligned} &0 && u \le \alpha, \\ &= (u-\alpha)/(\beta-\alpha) && \alpha < u \le \beta, \\ &= (\alpha-u)/(\beta-\alpha) && \alpha < u \le \gamma, \\ &= 0 && u > \gamma \end{aligned}\right\} \qquad (1.13)$$

The graphical representation of Triangular Membership Function is shown in Fig 1.9.

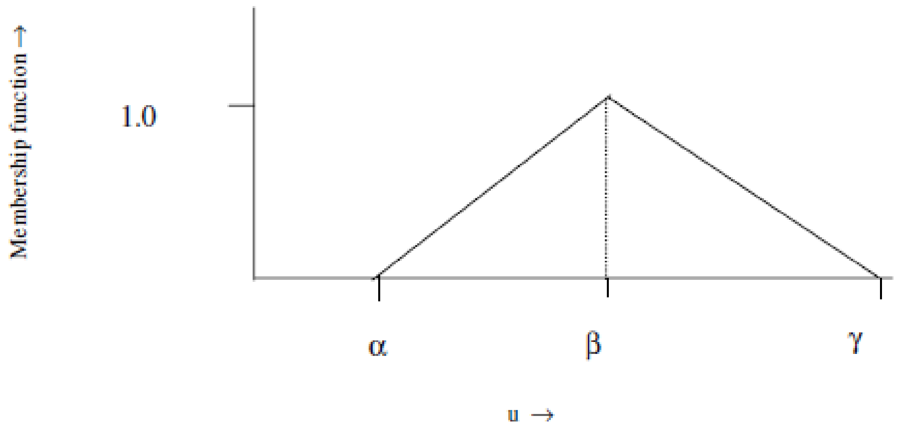


Figure 1.9 The Triangular membership function

1.5.5 The Π- Function

The Π-unction can be described as follows

$$\Pi(u;\alpha,\beta,\gamma,\delta) = \left.\begin{cases} 0 & u \le \alpha, \\ (u-\alpha)/(\beta-\alpha) & \alpha < u \le \beta, \\ 1 & \beta < u \le \gamma, \\ (\gamma-u)/(\delta-\gamma) & \gamma < u \le \delta, \\ 0 & u > \delta \end{cases}\right\} \quad (1.14)$$

The graphical representation of Π-function is shown in Fig 1.10.

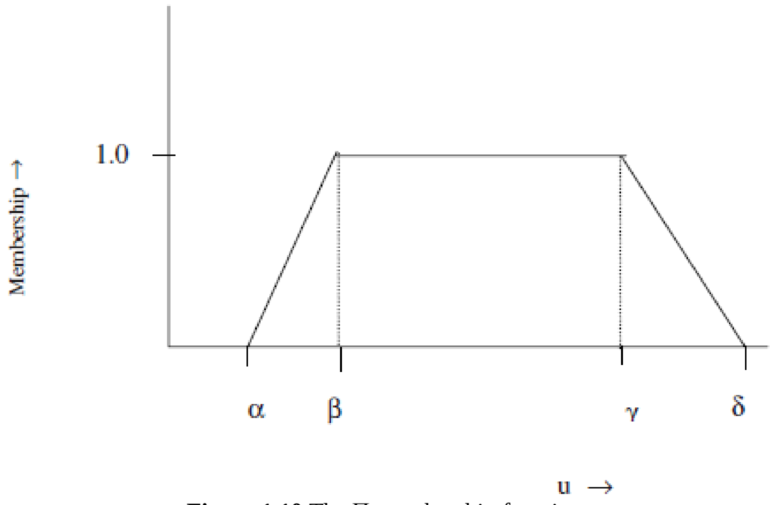


Figure 1.10 The Π-membership function

1.5.6 The Gaussian Membership Function

A Gaussian membership function is defined by

$$G(u; m, \sigma) = \exp[-\{(u-m)^2/\sqrt{2}\sigma\}^2] \qquad (1.15)$$

where the parameters m and σ control the centre and width of the membership function. A graphical plot of Gaussian membership function is shown in the figure (Fig 1.11) below.

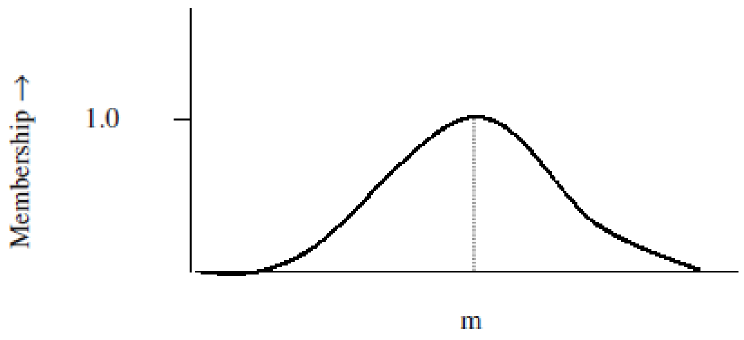


Figure 1.11 The Gaussian membership function

1.6 Explanation of Fuzzy Logic and Fuzzy Logic System

Fuzzy logic is a process which is used to determine the "fuzzy" decision making of a person by using the fuzzy sets. When a fuzzy logic is inputted as a program to the computer and a fuzzy logic system is established, the computer can accept all the information from the outside source and deal it more human-like fuzzy way.

Without using fuzzy logic a computer can deal with the same problems. But it is required high computation and high cost due to several iteration. If we use fuzzy logic and fuzzy logic controller then a computer can easily solve many real world problems. By using this fuzzy logic the computer can perform better than another computer which is used crisp logic.

A fuzzy logic system has three different main components: a) FUZZIFIER, b) LOGIC CONTROL CENTER, c) DEFUZZIFIER that are described below.

a) FUZZIFIER: A fuzzifier takes the number from outside the world and transforms them into fuzzy-input sets (Fig 1.12).

b) LOGIC CONTROL CENTER: By using the fuzzy-input set, a logic controller that uses rules, converts them into fuzzy-output sets (Fig 1.13).

c) DEFUZZIFIER: A defuzzifier takes the fuzzy outputs from the logic control center and converts it to again crisp output or numbers which indicate what action should be taken or decision should be made. (Fig 1.14).

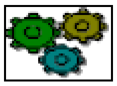


Figure 1.12 FUZZIFIER

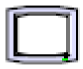


Figure 1.13 LOGIC CONTROL CENTER

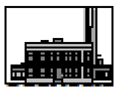


Figure 1.14 DEFUZZIFIER

A type-1 FLS is completely constructed by type-1 fuzzy sets. It contains four components-rulebase , fuzzifier, inference engine and defuzzifier, as shown in Fig 1.15.

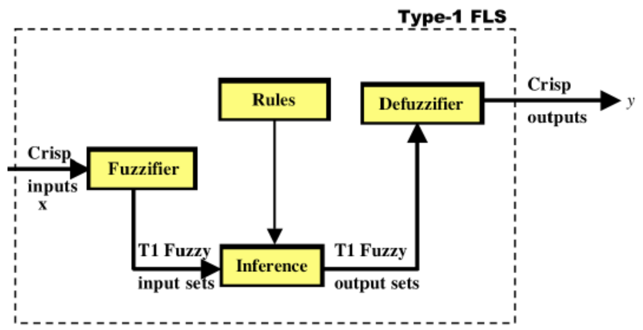


Figure 1.15 Type-1 Fuzzy Logic System

The rulebase is a collection of IF-THEN statements in the following form:

$$R^l\text{: IF } x_1 \text{ is } X_1^l \text{ and}\ldots\text{and } x_p \text{ is } X_p^l\text{, THEN } y \text{ is } Y^l$$

where X_i^l $(i = 1,\ldots,p)$ and Y^l are type-1 fuzzy sets in U_i and V respectively.

and $x = (x_1,\ldots,x_p) \in U_1 \times U_2 \times \ldots \times U_p \equiv U$ and $y \in V$ are linguistic variables.

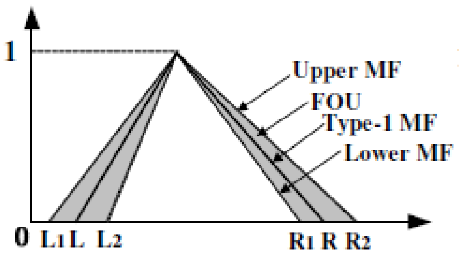
More specifically, x is the input and y is the output to the FLS. The IF-part of a rule is called its antecedent, and the THEN-part is called its consequent. All the fuzzy sets are associated with terms that appear in the antecedents or consequents of rules, and with the inputs to and output of the FLS. They are known as membership functions (MFs), which offer a measure of the degree of similarity of an element to the fuzzy set. For the type-1 fuzzy sets, the MFs are totally certain.

The fuzzifier performs a mapping from the crisp input $x = (x_1,\ldots,x_p)$ into fuzzy sets in U. In the fuzzy inference engine, fuzzy logic principles are generally used to combine the fuzzy IF-THEN rules in the fuzzy rule base into a mapping from the fuzzy sets in U to fuzzy sets in V. The defuzzifier implements a mapping from fuzzy sets in V to a crisp output $y \in V$.

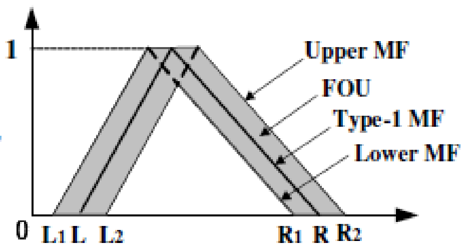
1.7 Type-2 Fuzzy Sets

Type-2 fuzzy sets (T2 FS) were first proposed by Zadeh in 1975 as an extension of Type-1 (T1-FS) fuzzy sets. A T1 FS fuzzy set has grade of membership that is crisp. Whereas a T2 FS fuzzy set has grade of membership which is fuzzy. It is called the “fuzzy fuzzy set”. Basically the membership function of T2 FS is also a T1 FS. All the T1 FS are embedded into the T2 FS. The T2 FSs and fuzzy logic systems (FLSs) have been gaining significant attentions in recent times. Particularly, people are paying attention in interval type-2 (IT2) FSs [Mendel 2001], whose memberships are intervals (instead of T1 FSs in a general T2 FS), for their simplicity and reduced the computational cost. IT2 FSs and FLSs have been used in many applications, including computing with words (CWW) [D. Wu and Mendel] linguistic data summarization modeling and control pattern recognition recommendation systems etc., and they often make obvious better performance than their T1 counterparts.

Examples of type-2 fuzzy sets are shown in Figure 1.16. The FOU is shown as the shaded region. It represents the uncertainties in the shape and position of the type-1 fuzzy set. The FOU is bounded by an upper MF and a lower MF, both of which are type-1 MFs. Consequently, the membership grade of each element in a type-2 fuzzy set is a type-1 fuzzy set [l; r], where l and r are membership grades on the lower and upper MFs. Type-2 sets are extremely suitable in circumstances where it is difficult to determine the exact MF for a fuzzy set; hence, they are useful for incorporating uncertainties.



(a) A type-2 fuzzy set developed by blurring the width of a triangular type-1 fuzzy set



(b) A type-2 fuzzy set developed by blurring the apex of a triangular type-1 fuzzy set

Figure 1.16 AType-2 Fuzzy sets

Type-2 fuzzy sets are challenging to understand and use because: (1) It is very difficult to draw the three-dimensional nature of type-2 fuzzy sets; (2) To effectively communicate about type-2 fuzzy sets; there is no simple collection of well-defined terms and to then be mathematically precise about them. (3) Sometimes terms do exist but have not been precisely defined. (4) Derivations of the formulas for the union, intersection, and complement, fuzziness, cardinality of type-2 fuzzy sets all rely on using Zadeh's Extension Principle which in itself is a difficult concept, especially for newcomers to FL, and is somewhat ad hoc, so that deriving things using it may be considered problematic. (5) The type-2 fuzzy sets are computationally more complicated than type-1 fuzzy sets.

1.7.1 Definition of Type-2 Fuzzy Sets

First we imagine a type-1 membership function that is shown in Fig. 1.17 by shifting all the points on the triangle either to the left or to the right and it is not necessary by the same amounts, it creates the blurred type-1 membership function as in Fig.1.18. Then, at a specific value of x, say x´, there is no longer a single value for the membership function u´ though the membership function takes on values wherever the vertical line intersects the blur. Those values need not all be weighted the same. Therefore we can assign an amplitude distribution to all of those points. Doing this for all x ∈ X, we create a three-dimensional (3D) membership function which is known as a type-2 membership function that characterizes the type-2 fuzzy set.

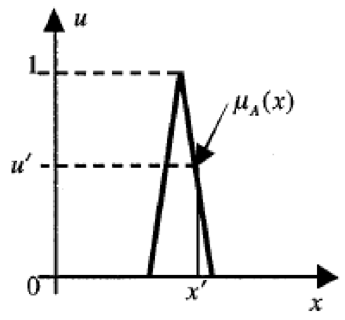


Figure 1.17 The Type-1 membership function

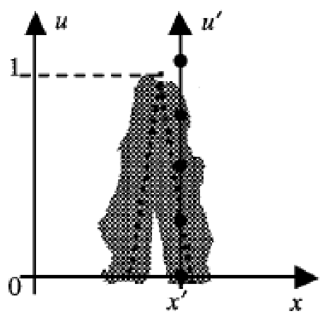


Figure 1.18 The blurred type-1 membership function

Definition 1.3 A type-2 fuzzy set, denoted by $\tilde{A}$, is characterized by a type-2 membership function $\mu_{\tilde{A}}(x,u)$, where $x \in X$ and $u \in J_x \subseteq [0,1]$. i.e.,

$$\tilde{A} = \{((x,u), \mu_{\tilde{A}}(x,u)) \mid \forall x \in X, \forall u \in J_x \subseteq [0,1]\} \qquad (1.16)$$

in which $0 \leq \mu_{\tilde{A}}(x,u) \leq 1$. $\tilde{A}$ can be also expressed as

$$\tilde{A} = \int_{x \in X} \int_{u \in Jx} \mu\tilde{A}(x, u) / (x, u), \quad J_x \subseteq [0,1] \qquad (1.17)$$

Here x is called the primary variable, has domain X and $u \in [0,1]$ is known as secondary variable, has domain $J_x \subseteq [0,1]$ at each $x \in X$; J_x is also called the support of the secondary membership function and the amplitude of $\mu_{\tilde{A}}(x,u)$ is called a secondary grade of $\tilde{x}$ which is equal to 1 for $\forall x \in X$ and $\forall u \in J_x \subseteq [0,1]$. And $\int$ denotes union over all admissible x and u. For discrete universe of discourse $\int$ is replaced by $\sum$.

In definition 1 the first restriction is $\forall u \in J_x \subseteq [0,1]$, which is consistent with the type-1 fuzzy set constraint $0 \leq \mu_{\tilde{A}}(x,u) \leq 1$. It indicates that when the uncertainties is disappeared then a T2 membership function (MF) must reduce to a T1 MF, in which case the variable u equals to $\mu_A(x)$ and $0 \leq \mu_A(x,u) \leq 1$. The second restriction $0 \leq \mu_{\tilde{A}}(x,u) \leq 1$, which is consistent with the fact that the amplitudes of the MF should lie between or be equal to 0 and 1. Fig 1.19 represents $\mu_{\tilde{A}}(x, u)$, where x and u are discrete. The value of x and u are x = {1, 2, 3, 4, 5}, u = {0.0, 0.2, 0.4, 0.6, 0.8}

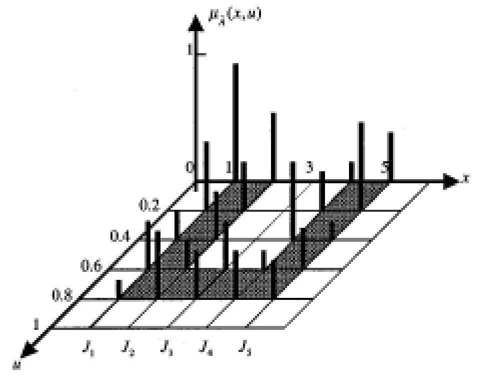


Figure 1.19 The Type-2 MF

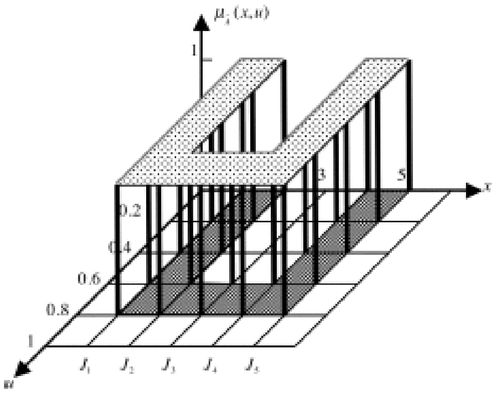


Figure 1.20 An IT2 MF for discrete universe of discourse

1.8 Interval Type-2 Fuzzy Sets (IT2 FSs)

Definition 1.4 When all the type-2 membership functions $\mu_{\tilde{A}}(x,u) = 1$ then $\tilde{A}$ will be the interval type-2 fuzzy set (IT2 FS).

Sometimes third dimension of general T2 FS gives no new information about the real world uncertainty. Hence it is required the IT2 FS. Basically interval type-2 fuzzy set is the special case of general type-2 fuzzy set. The third dimensional membership function of an IT2 FS is also an interval that is shown in the Fig 1.20. An IT2 FS can be expressed by the following equation (1.18).

$$\tilde{A} = \int_{x \in X} \int_{u \in Jx} 1/(x, u), \quad J_x \subseteq [0,1] \tag{1.18}$$

For each value of x, we consider x = x, the two dimensional plane whose are u and $\mu_{\tilde{A}}(x',u)$ is known as a vertical slice of $\mu_{\tilde{A}}(x,u)$. A secondary membership function is a vertical slice of $\mu_{\tilde{A}}(x,u)$. It is $\mu_{\tilde{A}}(x = x',u)$ for $x \in X$ and $\forall u \in J_x \subseteq [0,1]$, which is defined in the following equation (1.19).

$$\mu_{\tilde{A}}(x = x',u) \equiv \mu_{\tilde{A}}(x',u) = \int_{u \in J_{x'}} f_{x'}(u)/u \quad J_{x'} \subseteq [0,1] \tag{1.19}$$

where $0 \leq f_{x'}(u) \leq 1$ and $\forall x' \in X$, the prime notation $\mu_{\tilde{A}}(x')$ must be dropped and referred to $\mu_{\tilde{A}}(x)$ as a secondary membership function. It is a type-1 fuzzy set, which we can also refer to as a secondary set.

Again we consider the value of x and u are x = {1, 2, 3, 4, 5}, u = {0.0, 0.2, 0.4, 0.6, 0.8}. An interval type-2 membership function for discrete universe of discourse is shown in the figure (Fig 1.20). It has five vertical slices associated with it. The one at x = 2 is illustrated in Fig 1.21. The secondary membership function at x = 2 is $\mu_{\tilde{A}}(2) = 1/0.0 + 1/0.2 + 1/0.4 + 1/0.6 + 1/0.8$.

Based on the concept of the secondary sets, we can reinterpret an IT2 FS as the union of all secondary sets, by using equation 1.19; we can re-express Ã in a vertical-slice manner in equation 1.20.

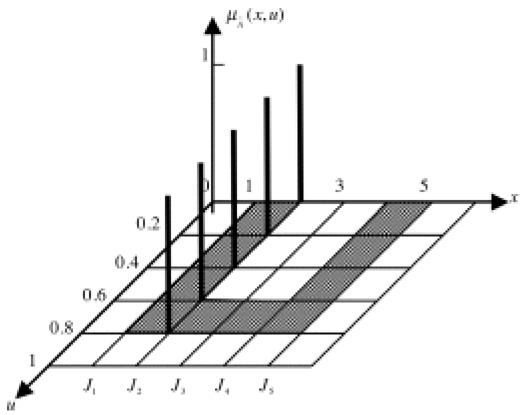


Figure 1.21 Example of a vertical slice of an IT2 MF illustrated in Fig. 1.19.

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid \forall x \in X\} \tag{1.20}$$

The expression of Ã can be written in different form by using the equation 1.21, which is given below

$$\tilde{A} = \int_{x \in X} \mu_{\tilde{A}}(x)/x = \int_{x \in X} \left[\int_{u \in J_x} f_x(u)/u\right]/x, \; J_x \subseteq [0,1] \tag{1.21}$$

The domain of a secondary membership function is called the primary membership of x. In equation 1.21, J_x is the primary membership of x, where $J_x \subseteq [0,1]$ for $\forall x \in X$. The amplitude of a secondary membership function is called a secondary grade. In equation 1.21 $f_x(u)$ is a secondary grade. In equation 1.16, $\mu_{\tilde{A}}(x,u) \mid \forall x \in X, \forall u \in J_x \subseteq [0,1]$ is a secondary grade. The secondary grades of an IT2 FS are all equal to 1, i.e., $f_x(u) = 1$. Then the expression of Ã can be written as

$$\tilde{A} = \int_{x \in X} \mu_{\tilde{A}}(x)/x = \int_{x \in X} [\int_{u \in J_x} 1/u]/x \ , \ J_x \subseteq [0,1] \tag{1.22}$$

If X and J_x are both discrete (discretization of continuous universes of discourse) then the right most part of the equation 1.21 can be written as

$$\begin{aligned} \tilde{A} &= \sum_{x \in X} [\sum_{u \in J_x} f_x(u)/u]/x \\ &= \sum_{i=1}^{N} [\sum_{u \in J_{x_i}} f_{x_i}(u)/u]/x_i \\ &= \sum_{i=1}^{M_1} [\sum_{u \in J_{x_i}} f_{x_1}(u_{1k})/u_{1k}]/x_1 + \ldots + \sum_{i=1}^{M_N} [\sum_{u \in J_{x_i}} f_{x_N}(u_{Nk})/u_{Nk}]/x_N \end{aligned} \tag{1.23}$$

Here "+" sign denotes the union operation. It is observed that x has been discretized into N values and at each of these values u has been discretized into M_i values. The discretization along each u_{ik} does not have to be the same. Therefore it is shown a different upper sum for each of the bracketed terms; however, if the discretization of each u_{ik} is the same, then $M_1 = M_2 = \ldots = M_N \equiv M$

In Fig 1.19 the union of the five secondary MFs at x = 1, 2, 3, 4, 5 is $\mu_{\tilde{A}}(x,u)$. It is observed that the primary memberships are $J_1 = J_2 = J_4 = J_5 = \{0.0, 0.2, 0.4, 0.6, 0.8\}$ and $J_3 = \{0.6, 0.8\}$. It is already included the values J_3 for which $\mu_{\tilde{A}}(x,u) \neq 0$. Each of the spikes in Fig. 1.19 represents $\mu_{\tilde{A}}(x,u)$ at a specific (x,u) - pair, and its amplitude of 1 is the secondary grade.

1.9 Footprint of Uncertainty (FOU)

Definition 1.5 In an interval type-2 fuzzy set Ã, the uncertainty in the primary memberships consists of a bounded region is called footprint of uncertainty (FOU). It is the union of all primary memberships, i.e.,

$$\text{FOU}(\tilde{A}) = \bigcup_{x \in X} J_x = \{(x,u) : u \in J_x \subseteq [0,1]\} \tag{1.24}$$

This is a vertical-slice representation of the FOU, because each of the primary memberships is a vertical slice.

The size of an FOU is directly related to the uncertainty that is conveyed by an IT2 FS. So, an FOU with more area is more uncertain than one with less area.

The upper membership function (UMF) and lower membership function (LMF) of Ã are two type-1 MFs that bound the FOU (Fig. 1.22). UMF (Ã) is associated with the upper bound of FOU (Ã) and is denoted $\overline{\mu}_{\tilde{A}}(x)$, $\forall$ x ∈ X, and LMF (Ã) is associated with the lower bound of FOU (Ã) and is denoted $\underline{\mu}_{\tilde{A}}(x)$, $\forall$ x ∈ X, we get

$$\text{UMF}(\tilde{A}) \equiv \overline{\mu}_{\tilde{A}}(x) = \overline{\text{FOU}(\tilde{A})} \qquad \forall x \in X \tag{1.25}$$

$$\text{LMF}(\tilde{A}) \equiv \underline{\mu}_{\tilde{A}}(x) = \underline{\text{FOU}(\tilde{A})} \qquad \forall x \in X \tag{1.26}$$

IT2 FS Ã is convex if both LMF (Ã) and UMF (Ã), which are T1 FSs, are convex over their respective supports. In general, the supports of LMF (Ã) and UMF (Ã) are different, and the support of LMF (Ã) is contained within the support of the UMF (Ã). All of our word FOUs satisfies this definition of convexity. The primary membership of J_x is an interval set, can be written as

$$J_x = [\underline{\mu}_{\tilde{A}}(x), \overline{\mu}_{\tilde{A}}(x)] \tag{1.27}$$

From set theory operation we get when U us continuous, the set is continuous and when U is discrete the set is discrete. It is denoted by $J_x\{\underline{\mu}_{\tilde{A}}(x), ..., \overline{\mu}_{\tilde{A}}(x)\}$. Using the equation 1.27, the FOU (Ã) in equation 1.24 can also be defined as

$$\text{FOU}(\tilde{A}) = \bigcup_{\forall x \in X} [\underline{\mu}_{\tilde{A}}(x), \overline{\mu}_{\tilde{A}}(x)] \tag{1.28}$$

Also we can describe an IT2 FS in a compact way by the following equation 1.29.

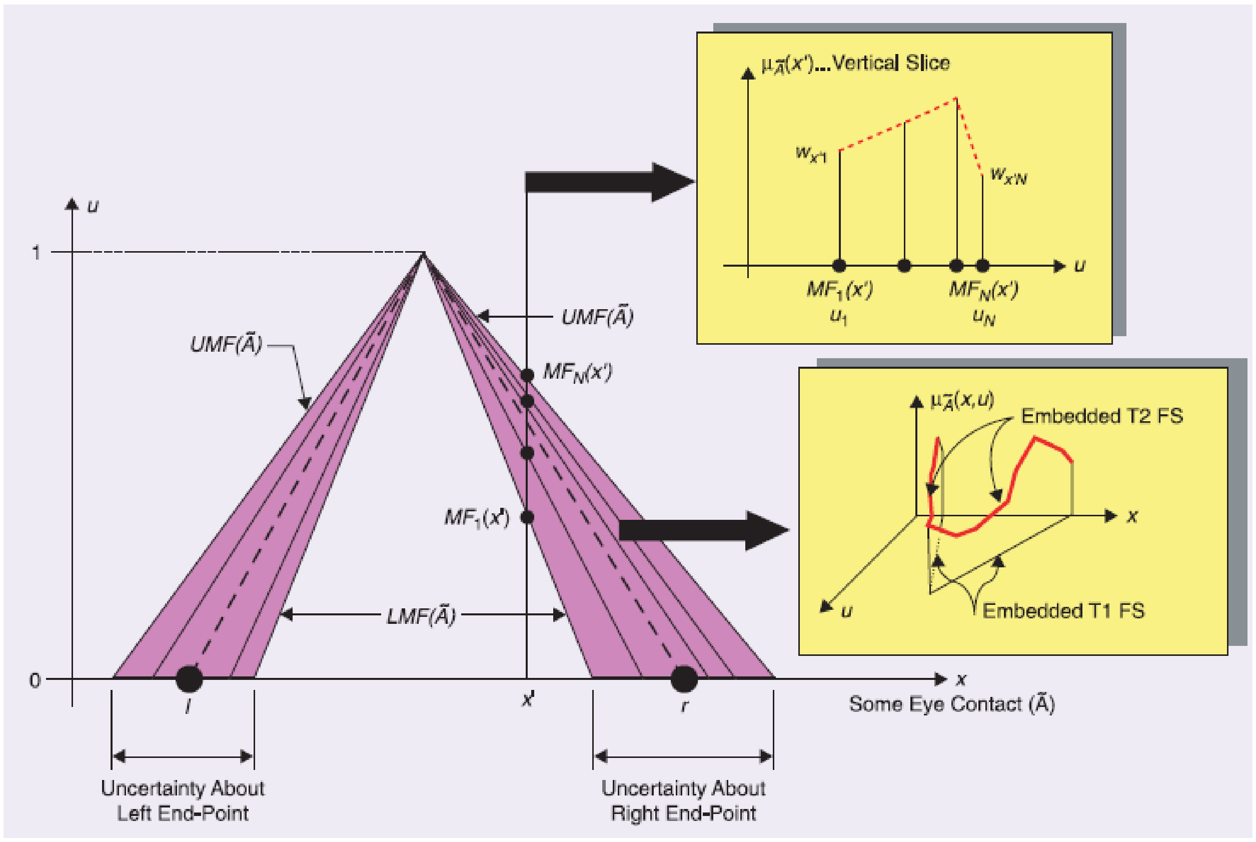


Figure 1.22 Footprint of Uncertainty (FOU (shaded)), LMF (dashed), UMF (solid) and Embedded T1 FS (wavy line)

$$\tilde{A} = 1/\ \text{FOU}\ (\tilde{A}) \qquad (1.29)$$

where this notation indicates that the secondary grade equals to 1 for all elements of FOU (Ã). Because all of the secondary grades of an IT2 FS equal to 1, these secondary grades express no useful information; hence, an IT2 FS is completely described by its FOU.

For continuous universes of discourse X and U, an embedded T1 FS A_e is

$$A_e = \int_{x \in X} u/x \qquad u \in J_x \qquad (1.30)$$

The set A_e is embedded in FOU (Ã). An example of A_e is given in Fig. 1.21. $\underline{\mu}_{\tilde{A}}(x)$ and $\overline{\mu}_{\tilde{A}}(x)$ are also the embedded type-1 FS in the FOU (Ã) for $\forall\ x \in X$. When the universes of discourse X and U are continuous then there are uncountable numbers of embedded T1 FSs in Ã. These sets are only used for the theoretical purposes and are not used for computational purposes. For discrete universes of discourse X and U, an embedded T1 FS A_e has N elements, one each from $J_{x1}, J_{x2}, \dots, J_{xN}$, namely $u_1, u_2, \dots, u_N$, that is,

$$A_e = \sum_{i=1}^{N} u_i/x_i \qquad u_i \in J_{xi} \qquad (1.31)$$

1.10 Brief idea on Interval Type-2 Fuzzy Logic System (IT2 FLS) & its components

Interval Type-2 Fuzzy Logic System (IT2 FLS) is similar to its Type-1 Fuzzy Logic System (T1 FLS). The major difference is that at least one of the FSs in the rulebase is an IT2 FS. Hence, the outputs of the inference engine are type-2 sets and a type-reducer is needed to convert them into a type-1 set before defuzzification can be carried out. Naturally IT2 FLS contains four components—rules, fuzzifier, inference engine and output processor—that are interconnected is shown in the figure 1.23. The output processor consists of type-reducer and defuzzifier.

The process is not quite simple as T1 FLS. The crisp inputs from the input sensors are first fuzzified into input interval type-2 fuzzy sets. Singleton fuzzification (where the input value is represented by precise crisp value) is usually used in interval type-2 FLS applications due to its simplicity and suitability for embedded processors and real-time applications. The input interval type-2 fuzzy sets then activate the inference engine and the rule base to produce output interval type-2 fuzzy sets. The interval type-2 FLS rules will remain the same as in Type-1 FLS (T1 FLS), but the antecedents and/or the consequents will be represented by interval type-2 fuzzy sets. The inference engine combines the fired rules and gives a mapping from input interval type-2 fuzzy sets to output interval type-2 fuzzy sets. The interval type-2 fuzzy outputs of the inference engine are then processed by the type-reducer, which combines the output sets and performs a centroid calculation that leads to type-1 fuzzy sets called the type-reduced sets. In the interval type-2 FLSs used so far, there are several ways to perform type-reduction. Karnik-Mendel (KM) algorithm or Enhanced Karnik-Mendel (EKM) algorithm are basically used for centroid calculation to perform type reduction. There are also various methods for centroid calculation, which are fast than KM or EKM and computational cost is less. Centroid calculation is now booming research area in type reduction at defuzzification process. After the type-reduction process, the type-reduced sets (or approximate type-reduced sets) are then defuzzified (by taking the average of the type reduced / approximated type-reduced set) to obtain crisp outputs that are sent to the actuators.

Rules are the heart of the IT2 FLS. They may be provided by experts or extracted from numerical data. In either case, the rules can be expressed as a collection of IF-THEN statements, e.g. IF the service is excellent and IF the food is delicious, THEN the tip is generous. The IF-part of a rule is its antecedent, and THEN-part of a rule is its consequent. Fuzzy Sets (FSs) are associated with terms that appear in the antecedents or consequents of rules, with the inputs to and output of the FLS. Membership Functions (MFs) are used to describe these FSs, and they can be either type-1 or type-2.

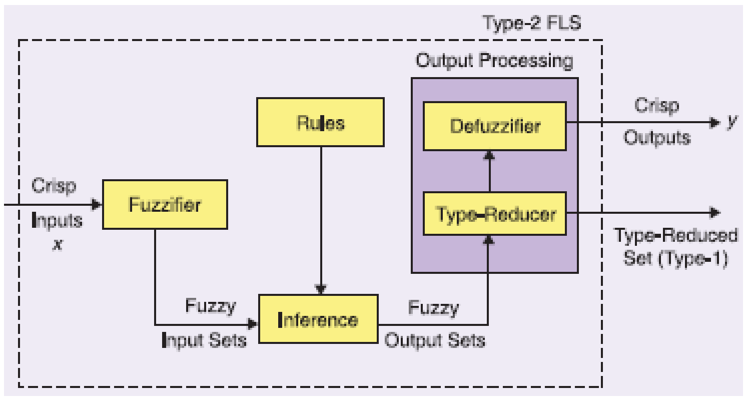


Figure 1.23 Type-2 Fuzzy Logic System

There are four ways in which uncertainty can occur in an FLS. These are (1) the words that are used in the antecedents or consequents of rules can mean different things to different people; (2) consequents is obtained by polling a group of experts will often be different for the same rule; because the experts will not necessarily be in agreement; (3) only noisy training data are available for tuning (optimizing) the parameters of an IT2 FLS; and (4) noisy measurements activate the IT2 FLS.

Once the rules have been established, an IT2 FLS can be viewed as a mapping from inputs to outputs (from “Crisp inputs” to “Crisp Outputs”). This mapping can be expressed quantitatively y = f(x). These kinds of IT2 FLSs are now enormously used in many engineering applications of Fuzzy Logic, such as in Fuzzy Logic Controllers (FLCs) and signal processors.

In the following subsections the operations in an interval singleton type-2 FLS are described in detail.

1.10.1 Fuzzification

Let us consider the rule base of a type-2 FLS consisting of N rules in the following form:

$$\tilde{R}^l: \text{IF } x_1 \text{ is } \tilde{X}_1^l \text{ and .... and } x_p \text{ is } \tilde{X}_p^l, \text{ THEN } y \text{ is } \tilde{Y}^l$$

where $\tilde{X}_i^l\,(i = 1,\dots,p)$ and $\tilde{Y}^l$ are type-2 fuzzy sets, and $x = (x_1,\dots,x_p)$ and y are linguistic variables.

The fuzzifier maps a crisp point $x = (x_1', x_2', \dots, x_p')$ into a type-2 fuzzy set $\tilde{A}_x$. Here it is focused on the type-2 singleton fuzzifier. This means $\mu_{\tilde{X}_i}(x_i) = 1/1$ when $x_i = x_i'$ and $\mu_{\tilde{X}_i}(x_i) = 1/0$ when $x_i \neq x_i'$, for all $i = 1, 2, \dots, p$.

1.10.2 Inference

The inference engine matches the fuzzy singletons with the fuzzy rules in the rule base. For computation the unions and intersections of type-2 sets, compositions of type-2 relations are required. Just as the sup-star composition is the backbone computation for a type-1 FLS, the extended sup-star composition is the backbone for a type-2 FLS.

The first step in the extended sup-star operation is to obtain the firing set $\Pi_{j=1}^{p} \mu_{\tilde{X}_j^i}(x_j) \equiv F^i(x)$ by performing the input and antecedent operations. As interval type-2 sets are only used and the meet operation is implemented by the product t-norm, the firing set is the following type-1 interval set:

$$F^i(x) = \left[\underline{f}^i(x), \overline{f}^i(x)\right] \equiv \left[\underline{f}^i, \overline{f}^i\right] \tag{1.32}$$

where $\underline{f}^i(x) = \underline{\mu}_{\tilde{X}_1^i}(x_1) * \underline{\mu}_{\tilde{X}_2^i}(x_2)$ and $\overline{f}^i(x) = \overline{\mu}_{\tilde{X}_1^i}(x_1) * \overline{\mu}_{\tilde{X}_2^i}(x_2)$. The term $\underline{\mu}_{\tilde{X}_j^i}(x_j)$ and $\overline{\mu}_{\tilde{X}_j^i}(x_j)$ are the lower and upper membership grades of $\mu_{\tilde{X}_j^i}(x_j)$ in interval type-2 fuzzy sets. Next, the firing set, $F^i(x)$, is combined with the consequent fuzzy set of the ith rule, $\mu_{\tilde{Y}^i}$, using the product t-norm to find out the fired output consequent sets. The combined output fuzzy set may then be obtained using the maximum t-conorm.

1.10.3 Defuzzification and Type-reduction

Defuzzification is the process of producing quantifiable result in crisp logic, given fuzzy sets and corresponding membership degrees.The main structural difference between type-1 FLS and type-2 FLS is that the latter must be type reduced before the defuzzifier can be used to generate a crisp output. This is because the output of the inference engine are generally type-2 fuzzy sets. The most commonly used type-reduction method is the Centroid Type Reduction, which may be expressed as

$$Y_{COS}(x) = \int_{y^1 \in Y^i} \cdots \int_{y^N \in Y^i} \int_{f^1 \in F^i(x)} \cdots \int_{f^N \in F^i(x)} \left[1 \Big/ \frac{\sum_{i=1}^{N} f^i y^i}{\sum_{i=1}^{N} f^i} \right] = [y_l, y_r] \tag{1.33}$$

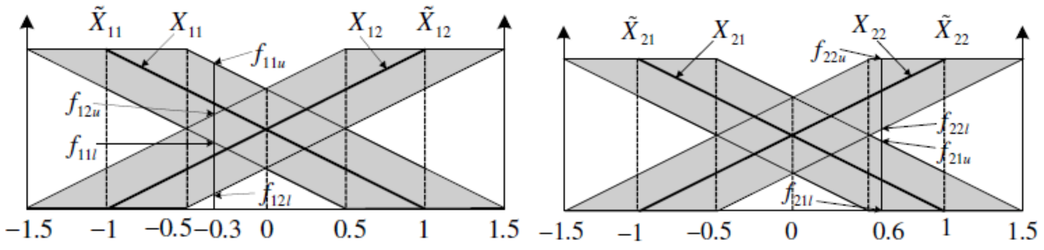
where $F^i(x) = \left[\underline{f}^i, \overline{f}^i\right]$ is the interval firing level of the ith rule, $Y^i = \left[y_l^i, y_r^i\right]$ is an interval type-1 set corresponding to the centroid of the interval type-2 consequent set $\tilde{Y}^i$:

$$C_{\tilde{Y}^i} = \int_{\theta_1 \in J_{x_1}} \cdots \int_{\theta_N \in J_{x_N}} 1 \Big/ \frac{\sum_{i=1}^{N} x_i \theta_i}{\sum_{i=1}^{N} \theta_i} = \left[y_l^i, y_r^i\right] \tag{1.34}$$

Equation (1.33) may be computed using the Karnik-Mendel algorithm that is completely described in section 2.4 (Chapter 2).

1.10.4 Example of a Type-2 FLS

In this section, the mathematical operations in a type-2 FLS are described using an example. Consider a baseline type-1 FLS that has two inputs (x_1 and x_2) and one output (y). It is assumed that each input domain consists of two type-1 MFs, shown as the dark thick lines in Figure 1.24.



(a) Input MFs of x_1 (b) Input MFs of x_2

Figure 1.24 The MFs of two FLSs

A type-2 FLS is obtained by equipping the four fuzzy sets used to partition the input domains of the baseline type-1 FLS with FOUs that is shown as the shaded areas in Figure 1.24. The rule base also has four rules assuming the following form:

$$\tilde{R}^{ij} : \text{IF } x_1 \text{ is } \tilde{X}_{1i} \text{ and .... and } x_2 \text{ is } \tilde{X}_{2j}, \text{ THEN } y \text{ is } \tilde{Y}_{ij} \text{ . } i, j = 1, 2$$

The complete rule base and the corresponding consequents are shown in Table 1.1.

Table 1.1 Rule base and consequents of the type-2 FLS

x_1 \ x_2	$\tilde{X}_{21}$	$\tilde{X}_{22}$
$\tilde{X}_{11}$	$\tilde{Y}_{11} = -1$	$\tilde{Y}_{12} = -0.5$
$\tilde{X}_{12}$	$\tilde{Y}_{21} = 0.5$	$\tilde{Y}_{22} = 1$

Consider an input vector x = (x_1, x_2) = (-0.3, 0.6). The firing strengths of the four type-2 input MFs are:

$$\tilde{f}_{11} = [0.4, 0.9] \qquad \tilde{f}_{12} = [0.1, 0.6]$$
$$\tilde{f}_{21} = [0, 0.45] \qquad \tilde{f}_{22} = [0.55, 1]$$

Thus the firing levels of the four rules are:

Rule No.:	Firing Strength	$\rightarrow$ Consequent
$\tilde{R}^{11}$:	$[0.4\times 0, 0.9\times 0.45] = [0, 0.405]$	$\rightarrow$ -1
$\tilde{R}^{12}$:	$[0.4\times 0.55, 0.9\times 1] = [0.22, 0.9]$	$\rightarrow$ -0.5
$\tilde{R}^{21}$:	$[0.1\times 0, 0.6\times 0.45] = [0, 0.27]$	$\rightarrow$ 0.5
$\tilde{R}^{22}$:	$[0.1\times 0.55, 0.6\times 1] = [0.055, 0.6]$	$\rightarrow$ 1

For the type-2 FLS, the bounds of the type-reduced interval type-1 set obtained using the Karnik-Mendel type-reducer are:

$$y_l = \frac{0.405\times(-1) + 0.22\times(-0.5) + 0\times 0.5 + 0.055\times 1}{0.405 + 0.22 + 0 + 0.055} = -0.6765$$

$$y_r = \frac{0\times(-1) + 0.22\times(-0.5) + 0\times 0.5 + 0.6\times 1}{0 + 0.22 + 0 + 0.6} = 0.5976$$

Here it is noticed the switch of membership grades for y_l occurs at $\tilde{R}^{11}$. That is, for consequent -1, the upper membership grade is used to calculate y_l. For consequents -0.5, 0.5 and 1, the lower membership grades are used. For y_r, the switch occurs after the second rule. That is, for consequent -1 and -0.5, the upper membership grades are used to calculate y_r and for consequents 0.5 and 1, the lower membership grades are employed.

Finally, the crisp output of the type-2 FLS, y_2 is calculated as:

$$y_2 = \frac{y_l + y_r}{2} = \frac{-0.6765 + 0.5976}{2} = 0.03945$$

1.11 Advantages of IT2 FLC over T1 FLC

The following are the advantages of IT2 FLC over T1 FLC.

1) IT2 fuzzy sets are capable to model and handle the linguistic and numerical uncertainties associated with the FLC and thus have the potential to produce a better performance when dealing with real world of uncertainties [Hagras 2007].

2) In type-2 fuzzy sets the uncertainty is represented in the footprint of uncertainty and allows us to cover the same range as type-1 fuzzy sets but with a smaller number of labels. This reduction of FLC rule base size will be greater when the number of the FLC inputs increases [Mendel 2001], [Hagras 2007].

3) A very large number type-1 fuzzy sets are used to describe each of the IT2 input and output set. This spontaneously allow for an exhaustive description of the analytical control surface as the addition of the extra levels of classification giving a much smoother control surface and response.

4) A type-2 FLC can model input – output relationship of higher complexity than its type-1 counterpart. As a result it provide better control response. The extra degree of freedom provided by foot of uncertainty allows a type-2 FLS to produce output that cannot be realized by type-1 with equal number of membership function. Moreover a type-2 fuzzy set can provide upswing to a similar type-1 negative or larger than unity membership grade [Wu 2010].

The computation of type-2 FLC is quite complex. Thus interval type-2 FLC is used which has much simplified computation and enable us to design FLC that operates in real time.

1.12 Wavy-Slice Representation of an IT2 FS

There are many diverse problem that involves IT2 fuzzy sets while considering set theoretic operations, centroid, uncertainty measures, similarity interference engine computation, linguistic weighted average etc. So a new type of representation known as Wavy-Slice Representation can be used to solve such above mentioned problems. It was first presented in Mendel and John

$$\tilde{A} = \int_{x \in X} \mu_{\tilde{A}}(x)/x \qquad (1.35)$$

(2002) for an arbitrary T2 FS.

Assume that the primary variable x of an IT2 FS is sampled at N values, $x_1, \ldots, x_N$, and at each of these values its primary memberships u_i are sampled at M_i values, $u_{i1}, \ldots, ui_{Mi}$. Then Ã is represented by equation (1.29). The footprint of uncertainty (FOU) of Ã can be written as

$$\text{FOU}(\tilde{A}) = \bigcup_{j=1}^{n_A} A_e^j = \{\underline{\mu}_{\tilde{A}}(x), \ldots, \overline{\mu}_{\tilde{A}}(x)\} \equiv [\underline{\mu}_{\tilde{A}}(x), \overline{\mu}_{\tilde{A}}(x)] \qquad (1.36)$$

The previous equation 1.36 represents the footprint of uncertainty FOU (Ã) as a union of all embedded T1 FS. This is the wavy-slice representation of an IT2 FS.

1.13 Set Theoretic operation of an IT2 FS

Here the Set Theoretic operation are described below which involves union, intersection and complement of an IT2 FS. Based on IF-THEN rules CWW (Computing With Word) engine uses these operations quite often.

First we consider two IT2 FS, Ã and $\tilde{B}$ (Fig 1.25(a)). The union operation (Fig 1.25(b)) is represented as $\tilde{A} \cup \tilde{B}$, which is another IT2 FS, with FOU ($\tilde{A} \cup \tilde{B}$). It can be expressed as

$$\tilde{A} \cup \tilde{B} = 1/\,\mathrm{FOU}(\tilde{A} \cup \tilde{B}) = 1/[\underline{\mu}_{\tilde{A}}(x) \vee \underline{\mu}_{\tilde{B}}(x), \overline{\mu}_{\tilde{A}}(x) \vee \overline{\mu}_{\tilde{B}}(x)] \tag{1.37}$$

where $\vee$ denotes the maximum operator.

The intersection operation (Fig 1.25(c)) is represented as $\tilde{A} \cap \tilde{B}$, which is another IT2 FS, with footprint of uncertainty FOU ($\tilde{A} \cap \tilde{B}$). It can be expressed as

$$\tilde{A} \cap \tilde{B} = 1/\,\mathrm{FOU}(\tilde{A} \cap \tilde{B}) = 1/[\underline{\mu}_{\tilde{A}}(x) \wedge \underline{\mu}_{\tilde{B}}(x), \overline{\mu}_{\tilde{A}}(x) \wedge \overline{\mu}_{\tilde{B}}(x)] \tag{1.38}$$

where $\wedge$ denotes the minimum operator.

The complement operation of an IT2 FS $\tilde{A}$ is $\tilde{A}^c$, which is also an IT2 FS, with FOU ($\tilde{A}^c$). The complement of the IT2 FS can be written as

$$\tilde{A}^c = 1/\,\mathrm{FOU}(\tilde{A}^c) = 1/[(1-\overline{\mu}_{\tilde{A}}(x)), (1-\underline{\mu}_{\tilde{A}}(x))] \tag{1.39}$$

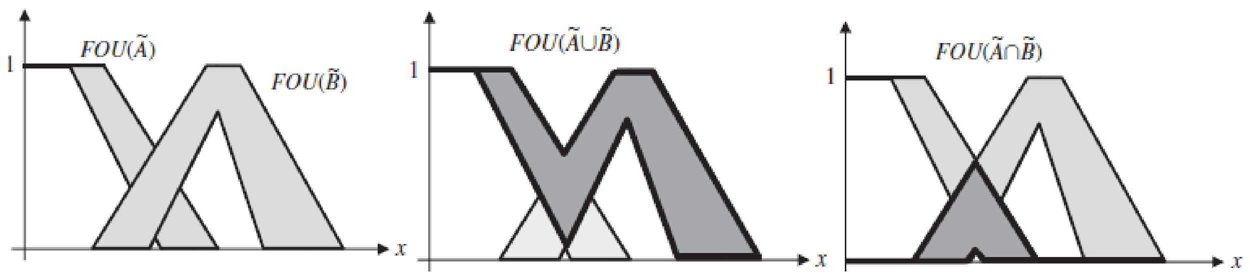


Figure 1.25 (a) Two IT2 FSs $\tilde{A}$ and $\tilde{B}$ (b) FOU of ($\tilde{A} \cup \tilde{B}$) and (c) FOU of ($\tilde{A} \cap \tilde{B}$)

1.14 Important concepts for an IT2 FS

Cardinality, fuzziness, variance, skewness, centroid, are all measures of uncertainty. Each of them is an interval and the length of the interval is an indicator of uncertainty. The centroid of an IT2 FS has been defined by Karnik and Mendel and is discussed widely in chapter 2 of the thesis. Here we will explain the other four concepts briefly.

1.14a Cardinality of an IT2 FS

In mathematics, the cardinality of a set is a measure of the "number of elements of the sets." Here in an IT2 FS cardinality means the measure of uncertainty. The cardinality is denoted by an interval and this interval indicates the extent of uncertainty. Larger is the interval, more is the uncertainty while smaller is the interval, less is the uncertainty. The alikeness between two IT2 FS may also be measure by this cardinality. It is used in to define a vector similarity measure for IT2 FSs.

Definitions of the cardinality of T1 FSs have been proposed by several authors, e.g. De Luca and Termini, Kaufmann, Gottwald, Zadeh, Blanchard, Klement, Wygralak etc. But here we consider two kinds of proposal. 1) The cardinality of a T1 FS could be a precise number. 2) The cardinality should be a fuzzy number. De Luca and Termini's have defined cardinality, also called the power of a T1 FS, is the sum of all the membership grades. It can be written as

$$P_A(A) = \frac{1}{N}\sum_{i=1}^{N} \mu_A(x_i) \tag{1.40}$$

ρ(A) can be defined as the average membership grade of A in its universe of discourse. It can also be seen that ρ(A) converges as N increases.

The cardinality can also be measured in T2 FS and IT2 FS. Szmidt and Kacprzyk have defined the interval cardinality for an IT2 FS Ã as

$$\begin{aligned} P_{\tilde{A}}(\tilde{A}) &= [\min_{\forall A_e} p_A(A_e), \max_{\forall A_e} p_A(A_e)] \\ &\equiv [p_A(\underline{\mu}_{\tilde{A}}), p_A(\overline{\mu}_{\tilde{A}})] \end{aligned} \tag{1.41}$$

The cardinality of an IT2 FS Ã is also defined as a union of all cardinalities of its embedded T1 FSs A_e, This can be written in equation 1.42.

$$P(\tilde{A}) = \bigcup_{\forall A_e} p(A_e) = [p_l, p_r] \tag{1.42}$$

Here $p_l = \min_{\forall A_e} p(A_e)$ and $p_r = \max_{\forall A_e} p(A_e)$ which is similar as equation 1.41.

1.14b Fuzziness of an IT2 FS

The fuzziness (entropy) of a T1 FS is used to calculate the amount of vagueness in it. A T1 FS C is most fuzzy when all its memberships equal 0.5. A T1 FS A is more fuzzy than a T1 FS B if A is nearer to such a C than B is. It is shown in the Fig 1.28.

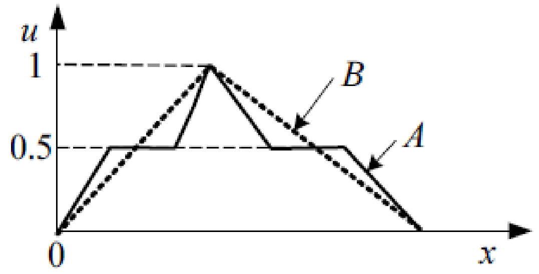


Figure 1.26 The Fuzziness of a T1 FS

In this figure A is more fuzzy than B because the memberships of A are closer to u = 0.5. There are several number of measures proposed for fuzziness. Kaufmann's index of fuzziness, which is defined by taking the Minkowski r-metric distance between A and the nearest crisp set A_{near}, i.e.

$$F_k(A) = \left[\sum_{i=1}^{N} \left| \mu_A(x_i) - \mu_{A_{near}}(x_i) \right|^r \right]^{\frac{1}{r}} \tag{1.43}$$

where A_{near} is defined as $\mu_{Anear} = 0$ if $\mu_A \leq ½$ and otherwise $\mu_{Anear} = 1$.

The fuzziness of an IT2 FS is defined by Burillo and Bustince in equation 1.44.

$$F_{\tilde{A}}(\tilde{A}) = \sum_{i=1}^{N} \left[\bar{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right] \tag{1.44}$$

The fuzziness $F_{\tilde{A}}$ of an interval type-2 fuzzy sets $\tilde{A}$ is a union of all embedded type-1 fuzzy sets A_e, i.e.,

$$F_{\tilde{A}} \equiv \bigcup_{\forall A_e} f(A_e) = [f_l, f_r] \tag{1.45}$$

where f_l and f_r are the minimum and maximum of fuzziness.

1.14c Variance of an IT2 FS

The variance of a T1 FS A measures its compactness. The variance of A is inversely proportional to its compactness, i.e. a smaller (larger) variance means A is more (less) compact. One popular definition of the variance of a T1 FS *A* is given by Carlsson and Fuller as *"the expected value of the squared deviations between the arithmetic mean and the endpoints of its level sets".* The variance of a T1 FS A is written as

$$v(A) = \frac{1}{N} \sum_{i=1}^{N} [x_i - c(A)]^2 \mu_A(x_i) \tag{1.46}$$

where c(A) is the centroid of T1 FS. Now the variance of an IT2 FS is the union of all the variance of embedded T1 FS. The variance of an IT2 FS is defined as

$$V_{\tilde{A}} \equiv \bigcup_{\forall A_e} v(A_e) = \bigcup_{\forall A_e} \left[\frac{1}{N} \sum_{i=1}^{N} [x_i - c(A_e)]^2 \mu_{A_e}(x_i) \right] \tag{1.47}$$

1.14d Skewness of an IT2 FS

The skewness of a T1 FS A, s(A), is an indicator of its symmetry. s(A) is smaller than zero when A skews to the right, is larger than zero when A skews to the left, and is equal to zero when A is symmetrical. In T1 FS skewness is defined as

$$s(A) = \left[\frac{1}{N} \sum_{i=1}^{N} [x_i - c(A)]^3 \mu_A(x_i) \right] \tag{1.48}$$

Now the skewness of an IT2 FS is the union of all the skewness of embedded T1 FS. The skewness of an IT2 FS is defined as

$$S_{\tilde{A}} \equiv \bigcup_{\forall A_e} s(A)_e = \bigcup_{\forall A_e} \left[\frac{1}{N} \sum_{i=1}^{N} [x_i - c(A_e)]^3 \mu_{A_e}(x_i) \right] \tag{1.49}$$

1.15 Comparison of T1, T2 and IT2 FSs

To compare the three types of fuzzy sets, we need to focus on the way they represent the degrees of membership. The same input p is applied to the three different types of fuzzy set as shown in Fig (1.27). Type-1 fuzzy set in Fig (1.27(a)), interval Type-2 fuzzy set in Fig (1.27(b)) and general Type-2 fuzzy set in Fig(1.27(c)) producing a degree of membership which is precisely based on type of fuzzy set. The amount of uncertainty (and the distribution) that is related with the degree is shown in color in Fig (1.27) and is enlightened in Figure (1.28). It displays the secondary Membership Functions (MFs) (third dimension) of the type-1 fuzzy set (Figure (1.28(a)), the interval type-2 fuzzy set (Figure (1.28(b)) and the general type-2 fuzzy set (Figure (1.28(c)) as induced by the same input p as shown in Figure (1.27). It should be illustrious that Figure (1.28) is envisaging the y-z plane.

As shown in Figure (1.28), the secondary MF in type-1 fuzzy sets has only one value in its domain "a" in Figure (1.28(a)) corresponding to the primary membership value at which the secondary grade equals 1. Hence, in type-1 fuzzy sets, for each p value, there is no uncertainty related with the primary membership value [Mendel 2001]. In interval type-2 fuzzy sets as displayed in Figure (1.28(b)), there is maximum uncertainty associated with the secondary membership function as the primary membership is

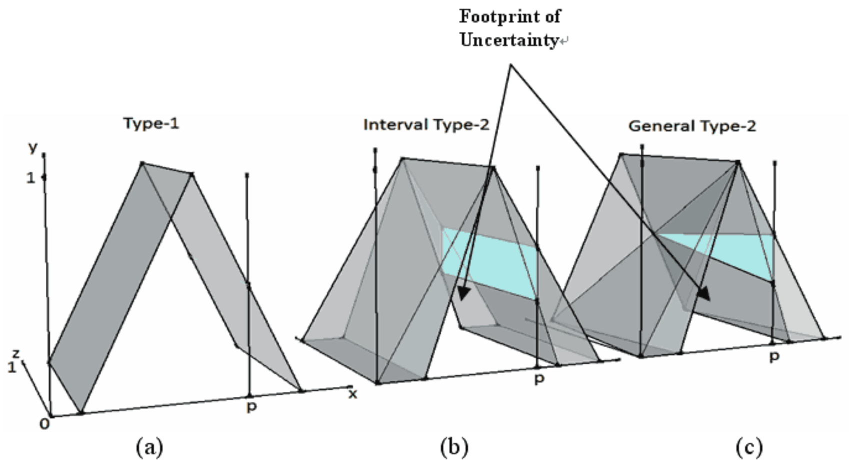


Figure 1.27 (a) Type-1 FS. (b) IT2 FS. (c) General Type-2 FS.

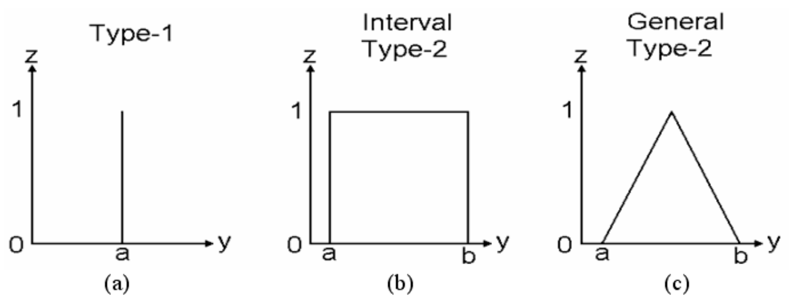


Figure 1.28 A view of the secondary MFs induced by an input p for (a) Type-1 FS. (b) Interval Type-2 FS and (c) General Type-2 FS.

taking values within the interval [a, b], where each point in this interval is having an associated secondary membership of 1. In general type-2 fuzzy sets as shown in Figure (1.28(c)), the uncertainty (represented by the secondary membership function) can be demonstrated with any degree between type-1 and interval type-2 fuzzy sets, for example by the triangular secondary membership function shown in Figure (1.28(c)). Overall, it can be noted that general type-2 fuzzy sets can model the uncertainty in the third dimension from nearly no uncertainty (i.e. type-1) to maximum (i.e. interval type-2, where the uncertainty is equally spread in the third dimension).

1.16 Conclusion

Type 1 FLCs have been applied to data with great success to different applications. However for many real-world environment and application, there is a need to cope with large amount of uncertainties. The traditional type 1 FLCs that use crisp type-1 fuzzy set cannot directly handle such uncertainties. While it has been seen that with the increase in the level of fuzziness and uncertainty, IT2 FLC can provide a powerful paradigm. It has also been seen in various application that the IT2 FLCs have given very well and smooth control responses that have outperformed their T1 counterpart.

In the world of latest technology where automation is the new boom, fuzzy control system has a vast range of application. IT2 FLCs, find several application in diverse field. It includes industrial application, business application and many more. Majority of these applications involve challenging environment which are often equivocating. IT2 FLCs is capable to provide the best possible solution to such cases.

Thus the IT2 FLC is credited with being an adequate methodology for designing robust controllers that are able to deliver satisfactory performance in the face of uncertainty and imprecision.

References

[1] L. A. Zadeh, “The concept of a linguistic variable and its application to approximate reasoning-1,” Inf. Sci., vol. 8, pp. 199–249, 1975.

[2] J. M. Mendel, Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions. Upper Saddle River, NJ: Prentice–Hall, 2001.

[3] D.Wu and J.M. Mendel, “Computing with words for hierarchical decision making applied to evaluating a weapon system,” IEEE Trans. Fuzzy Syst., vol. 18, no. 3, pp. 441–460, Jun. 2010.

[4] M. Gehrke, C. Walker, and E. Walker, “Some comments on interval valued fuzzy sets,” Int. J. Intell. Syst., vol. 11, pp. 751–759, 1996.

[5] J. M. Mendel, “Type-2 fuzzy sets: some questions and answers,” IEEE Connections, Newsletter of the IEEE Neural Networks Society, vol. 1, pp. 10–13, Aug. 2003.

[6] J. M. Mendel, “Type-2 fuzzy sets and systems: An overview,” IEEE Computational Intelligence Magazine, vol. 2, pp. 20–29, February 2007a.

[7] J. M. Mendel and R. I. John, “Type-2 fuzzy sets made simple,” IEEE Trans. on Fuzzy Systems, vol. 10, pp. 117–127, April 2002.

[8] J. M. Mendel, R. I. John, and F. Liu, “Interval type-2 fuzzy logic systems made simple,” IEEE Trans. on Fuzzy Systems, vol. 14, pp. 808–821, December 2006.

[9] F. Liu and J. M. Mendel, “Encoding words into interval type-2 fuzzy sets using an Interval Approach,” IEEE Trans. on Fuzzy Systems, vol. 16, pp 1503–1521, December 2008b.

[10] D. Dubois and H. Prade, “Fuzzy cardinality and the modeling of imprecise quantification,” Fuzzy Sets and Systems, vol. 16, pp. 199–230, 1985.

[11] A. De Luca and S. Termini, “A definition of non-probabilistic entropy in the setting of fuzzy sets theory,” Information and Computation, vol. 20, pp. 301–312, 1972.

[12] G. J. Klir and B. Yuan, Fuzzy Sets and Fuzzy Logic: Theory and Applications. Upper Saddle River, NJ: Prentice-Hall, 1995.

[13] J. M. Mendel and H. Wu, “Type-2 fuzzistics for non-symmetric interval type-2 fuzzy sets: Forward problems,” IEEE Trans. on Fuzzy Systems, vol. 15, pp. 916–930, October 2007b.

[14] L.-X. Wang, A Course in Fuzzy Systems and Control, Upper Saddle River, NJ: Prentice-Hall PTR, 1997.

[15] L.-X. Wang and J. M. Mendel, “Fuzzy basis functions, universal approximation, and orthogonal least-squares learning,” IEEE Trans. on Neural Networks, vol. 3, pp. 807–813, September 1992a.

[16] D. Wu and J. M. Mendel, “Uncertainty measures for interval type-2 fuzzy sets,” Information Sciences, vol. 177, pp. 5378–5393, 2007b.

[17] H. Wu and J. M. Mendel, “Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems,” IEEE Trans. on Fuzzy Systems, vol. 10, no. 5, pp. 622–639, 2002.

[18] M. Wygralak, “A new approach to the fuzzy cardinality of finite fuzzy sets,” Bulletin forStudies and Exchange of Fuzziness and its Applications, vol. 15, pp. 72–75, 1983.

[19] L. A. Zadeh, “Possibility theory and soft data analysis,” in L. Cobb and R. M. Thrall (Eds.), Mathematical Frontiers of the Social and Policy Sciences, Boulder, CO: Westview Press, pp. 69–129, 1981.

[20] E. P. Klement, “On the cardinality of fuzzy sets,” in Proceedings of the Sixth European Meeting on Cybernetics and Systems Research, Vienna, 1982, pp. 701–704.

[21] S. Coupland and R. I. John, “Geometric type-1 and type-2 fuzzy logic systems,” IEEE Trans. Fuzzy Syst., vol. 15, no. 1, pp. 3–15, Feb. 2007.

[22] H. Hagras, “Type-2 FLCs: A new generation of fuzzy controllers,” IEEE Computat. Intell. Mag., vol. 2, no. 1, pp. 30–43, Feb. 2007.

Chapter 2

Type-2 Centroids

The chapter provides an introduction of continuous and discrete version of centroid of a T2 FS. It gives an idea of iterative KM algorithm for type reduction in IT2 FS. This chapter also illustrates the three categories of methods to reduce their computational cost for finding centroid of an IT2 FS at type-reduction process. The first category consists of five enhancements to the KM algorithms, which are the most popular type-reduction algorithms to date. The second category consists of eleven alternative type-reducers, which have closed-form representations and, hence, are more convenient for analysis. The third category consists of a simplified structure for IT2 FLSs, which can be combined with any algorithms in the first or second category for further computational cost reduction.

2.1 Introduction

An interval Type-2 Fuzzy Set (IT2 FS) is today the most widely used Type-2 Fuzzy Set (T2 FS) because it is computationally simple to use. When such FSs are used in a rule-based Fuzzy Logic System (FLS), the result is an IT2 FLS. In such a FLS, fired-rule output sets are also IT2 FSs, and to go from such sets to a number one must perform two successive operations, type-reduction and defuzzification. Type-reduction maps the output T2 FS into a T1 FS and defuzzification converts that T1 FS into a number.

Type-reduction methods were developed by Karnik and Mendel. When they are applied to a general T2 FS they require an astronomical number of computations. When they are applied to an IT2 FS, they require a very small number of computations, which is one of the major reasons that IT2 FLSs have received attention whereas general T2 FLSs have not. Even for IT2 FLSs there can be many different kinds of type-reduction. The once that have been developed so far all extend a T1 centroid calculation to T2 FSs, so that if all sources of uncertainty disappear the output of an IT2 FLS reduces to that of an T1 FLS. So, computing the centroid of an IT2 FS plays a central role in IT2 FLSs.

The centroid of an IT2 FS also provides a measure of the uncertainty of an IT2 FS, and more recently has been the basis for going from data collected from a group of subjects (about an interval that they associate with the meaning of a word) to the Footprint of uncertainty(FOU) of an IT2 FS that models that word. One of the key points for finding alternative methods is to deal with minimizing the amount of arithmetic and logic operations. Hence, alternative methods for computing type reduction that provide both precision and fast computing times are desirable.

2.2 Brief discussion on Continuous version of the Centroid

Given an IT2 FS Ã which is defined on a universal set $X \in$ [illegible] , with membership function $\mu_{\tilde{A}}$ (x), x ∈ X. Its centroid (if it exists) c (Ã) is a closed interval [cl , cr] in the classical sense of mathematics, i.e.,

$$c\,(\tilde{A}) \equiv [c_l, c_r] \tag{2.1}$$

where c_r and c_l are respectively the maximum and minimum of all centroids of the embedded type-1 fuzzy sets (T1 FSs) A_e in the footprint of uncertainty (FOU) of type-2 fuzzy set (T2 FS), Ã (Fig 2.1(a)). Mendel has defined the continuous version for c_l and c_r of IT2 FS Ã as

$$\begin{aligned} c_l &= \min_{l \in R} \operatorname{centroid}(A_e(l)) \\ c_r &= \max_{r \in R} \operatorname{centroid}(A_e(r)) \end{aligned} \tag{2.2}$$

where

$$\text{centroid}(A_e(l)) = \frac{\int_{-\infty}^{+\infty} x\mu_{A_e(l)}(x)dx}{\int_{-\infty}^{+\infty} \mu_{A_e(l)}(x)dx} \tag{2.3}$$

$$\text{or, } \text{centroid}(A_e(l)) = \frac{\int_{-\infty}^{l} x\overline{\mu}_{\tilde{A}}(x)dx + \int_{l}^{+\infty} x\underline{\mu}_{\tilde{A}}(x)dx}{\int_{-\infty}^{l} \overline{\mu}_{\tilde{A}}(x)dx + \int_{l}^{+\infty} \underline{\mu}_{\tilde{A}}(x)dx} \tag{2.4}$$

$$\text{and } \text{centroid}(A_e(r)) = \frac{\int_{-\infty}^{+\infty} x\mu_{A_e(r)}(x)dx}{\int_{-\infty}^{+\infty} \mu_{A_e(r)}(x)dx} \tag{2.5}$$

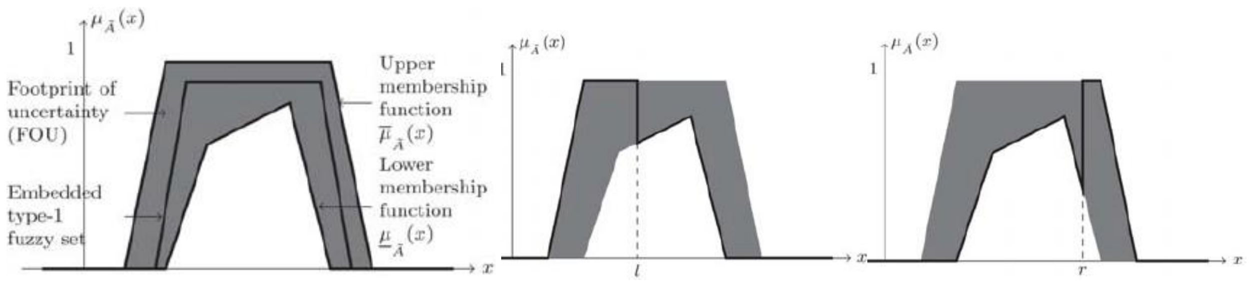
$$\text{or, } \text{centroid}(A_e(r)) = \frac{\int_{-\infty}^{r} x\underline{\mu}_{\tilde{A}}(x)dx + \int_{r}^{+\infty} x\overline{\mu}_{\tilde{A}}(x)dx}{\int_{-\infty}^{r} \underline{\mu}_{\tilde{A}}(x)dx + \int_{r}^{+\infty} \overline{\mu}_{\tilde{A}}(x)dx} \tag{2.6}$$

and where $A_e(l)$ and $A_e(r)$ denote embedded type-1 fuzzy sets for which:

$$\begin{aligned} \mu_{A_e(l)}(x) &= \overline{\mu}_{\tilde{A}}(x), \quad \text{if } x \le l \\ &= \underline{\mu}_{\tilde{A}}(x), \quad \text{if } x > l \end{aligned} \tag{2.7}$$

$$\begin{aligned} \mu_{A_e(r)}(x) &= \underline{\mu}_{\tilde{A}}(x), \quad \text{if } x \le r \\ &= \overline{\mu}_{\tilde{A}}(x), \quad \text{if } x > r \end{aligned} \tag{2.8}$$

According to Mendel, l, r ∈ X are switch points, i.e., values of x at which $\mu_{Ae(l)}(x)$ and $\mu_{Ae(r)}(x)$switch from $\overline{\mu}_{\tilde{A}}(x)$ to $\underline{\mu}_{\tilde{A}}(x)$ (or vice versa). $\overline{\mu}_{\tilde{A}}(x)$ and $\underline{\mu}_{\tilde{A}}(x)$ are the upper membership function and lower membership function of Ã (Figure 2.1(b) and Figure 2.1(c)).


$\mu_{\tilde A}(x)$
1
Footprint of uncertainty (FOU)
Embedded type-1 fuzzy set
Upper membership function $\bar{\mu}_{\tilde A}(x)$
Lower membership function $\underline{\mu}_{\tilde A}(x)$
x
l
r

(a) Membership function of an interval type-2 fuzzy set (b) Interpretation of the switch point *l* .(c) Interpretation of the switch point *r*.

Figure 2.1 Continuous version of the Centroid

2.2.1 Non-existence of the Centroid

Mendel et al. [2007] has studied properties of (2.4) and (2.6) assuming existence of the centroid, that is, convergence of the integrals that define it. However, this is not always true and there are some IT2 FS for which (2.4) and (2.6) do not exist in the sense that they are not finite. One example is the following.

Example 1 Let Ã be an IT2 FS which is defined on a universal set $X \in \mathbb{R}$, with membership function (MF), $\mu_{\tilde A}$ (x), x ∈ X. The upper membership function (UMF) and lower membership function (LMF) of Ã is defined as $\bar{\mu}_{\tilde A}(x)$ and $\underline{\mu}_{\tilde A}(x)$ respectively.

$$\bar{\mu}_{\tilde A}(x) = \frac{1}{1+x^2} \quad \text{and} \quad \underline{\mu}_{\tilde A}(x) = \frac{1}{2}\left(\frac{1}{1+x^2}\right)$$

Then the switching point "l" is defined as

$$\text{centroid}(A_e(l)) = \lim_{t\to+\infty} \frac{\int_{-t}^{l} x\bar{\mu}_{\tilde A}(x)dx + \int_{l}^{+t} x\underline{\mu}_{\tilde A}(x)dx}{\int_{-t}^{l} \bar{\mu}_{\tilde A}(x)dx + \int_{l}^{+t} \underline{\mu}_{\tilde A}(x)dx}$$

$$\Rightarrow \text{centroid}(A_e(l)) = \lim_{t\to+\infty} \frac{\int_{-t}^{l} \frac{x}{1+x^2}dx + \frac{1}{2}\int_{l}^{+t} \frac{x}{1+x^2}dx}{\int_{-t}^{l} \frac{1}{1+x^2}dx + \frac{1}{2}\int_{l}^{+t} \frac{x}{1+x^2}dx}$$

$$\Rightarrow \text{centroid}(A_e(l)) = \lim_{t\to+\infty} \frac{\frac{1}{4}\ln\left(\frac{1+l^2}{1+t^2}\right)}{\frac{1}{2}\arctan(l) + \frac{3}{2}\arctan(t)}$$

but the denominator $\frac{1}{2}\arctan(l) + \frac{3}{4}\arctan(t) \longrightarrow \frac{1}{2}\arctan(l) + \frac{3}{4}\pi$

and the numerator $\frac{1}{4}\ln\left(\frac{1+l^2}{1+t^2}\right) \rightarrow -\infty$

if $t \rightarrow +\infty$, Then the centroid $A_e(l) \rightarrow -\infty$ and it does not exists. It can also be shown that the centroid $A_e(r)$ (switching point "r") $\rightarrow +\infty$, which also does not exists. In this case equation 2.2 does not make sense. The Non-existence of the centroid is illustrated in Fig 2.2.

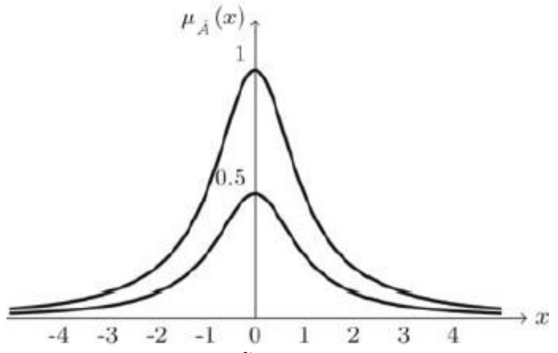


Figure 2.2 MF of an IT2 FS Ã, which does not have any centroid

Here $\overline{\mu}_{\tilde{A}}(x) = 0.5(1+x_2)$ and $\underline{\mu}_{\tilde{A}}(x) = 1/(1+x_2)$ $\forall x \in X = $

2.3 Brief discussion on Discrete version of the Centroid

Discrete version of the centroid which is more important than its continuous version was first illustrated by Karnik and Mendel. They demonstrated that c_l and c_r can be computed from the lower and upper membership functions of Ã as follows:

$$c_l = \min_{L\in N} \text{centroid}(A_e(L))$$
$$c_r = \max_{R\in N} \text{centroid}(A_e(R)) \qquad (2.9)$$

where

$$\text{centroid}(A_e(L)) = \frac{\sum_{i=1}^{L} x_i \overline{\mu}_{\tilde{A}}(x_i) + \sum_{i=L+1}^{N} x_i \underline{\mu}_{\tilde{A}}(x_i)}{\sum_{i=1}^{L} \overline{\mu}_{\tilde{A}}(x_i) + \sum_{i=L+1}^{N} \underline{\mu}_{\tilde{A}}(x_i)} \qquad (2.10)$$

and $$\text{centroid}(A_e(R)) = \frac{\sum_{i=1}^{R} x_i \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=R+1}^{N} x_i \overline{\mu}_{\tilde{A}}(x_i)}{\sum_{i=1}^{R} \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=R+1}^{N} \overline{\mu}_{\tilde{A}}(x_i)} \tag{2.11}$$

and where $L \in \mathbb{N}$ is the switch point that marks the change from $\underline{\mu}_{\tilde{A}}$ to $\overline{\mu}_{\tilde{A}}$ (Figure 2.3(b)), and $R \in \mathbb{N}$ is the switch point that marks the change from $\overline{\mu}_{\tilde{A}}$ to $\underline{\mu}_{\tilde{A}}$ (Figure 2.3(c)). $N \in \mathbb{N}$ is the number of discrete points on which the x-domain of Ã has been discretized. It is assumed that in equation (2.10) and (2.11) $x_1 < x_2 < \ldots < x_N$, in which x_1 denotes the smallest sampled value of x and x_N denotes the largest sampled value of x [3].

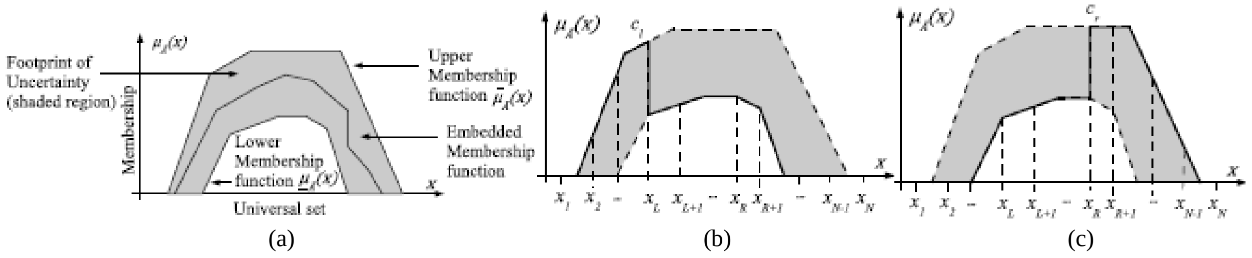

Footprint of Uncertainty (shaded region)
$\mu_{\tilde{A}}(x)$
Membership
Upper Membership function $\overline{\mu}_{\tilde{A}}(x)$
Embedded Membership function
Lower Membership function $\underline{\mu}_{\tilde{A}}(x)$
Universal set
x
c_l
c_r
x_1 x_2 x_L x_{L+1} x_R x_{R+1} x_{N-1} x_N
(a)
(b)
(c)

Figure 2.3 (a) IT2 FS (b) c_l and its interpretation (c) c_r and its interpretation

2.3.1 Deduction of a General Expression

A special property of the discrete version of the centroid was first observed by O. Salazar, J. Soriano, and H. Serrano [4]. We can rewrite the expression (2.11). Let j = N-1-i then we will have the following:

1. if $1 \le i \le R$ then $1 \le N+1-j \le R$, and hence $N-R+1 \le j \le N$.

2. if $R+1 \le i \le N$ then $R+1 \le N+1-j \le N$, and hence $1 \le j \le N-R$

Therefore equation (2.11) can be written as

$$\text{centroid}(A_e(R)) = \frac{\sum_{j=N-R+1}^{N} y_j \underline{\mu}_{\tilde{A}}(y_j) + \sum_{j=1}^{N-R} y_j \overline{\mu}_{\tilde{A}}(y_j)}{\sum_{j=N-R+1}^{N} \underline{\mu}_{\tilde{A}}(y_j) + \sum_{j=1}^{N-R} \overline{\mu}_{\tilde{A}}(y_j)}$$

$$\Rightarrow \text{centroid}(A_e(R)) = \frac{\sum_{j=1}^{N-R} y_j \bar{\mu}_{\tilde{A}}(y_j) + \sum_{j=N-R+1}^{N} y_j \underline{\mu}_{\tilde{A}}(y_j)}{\sum_{j=1}^{N-R} \bar{\mu}_{\tilde{A}}(y_j) + \sum_{j=N-R+1}^{N} \underline{\mu}_{\tilde{A}}(y_j)}$$

$$\Rightarrow \text{centroid}(A_e(R)) = \frac{\sum_{j=1}^{L'} y_j \bar{\mu}_{\tilde{A}}(y_j) + \sum_{j=L'+1}^{N} y_j \underline{\mu}_{\tilde{A}}(y_j)}{\sum_{j=1}^{L'} \bar{\mu}_{\tilde{A}}(y_j) + \sum_{j=L'+1}^{N} \underline{\mu}_{\tilde{A}}(y_j)} \qquad (2.12)$$

where

$$y_j = x_{N+1-j},\ 1 \le j \le N \qquad (2.13)$$

and

$$L' = N\text{-}R \qquad (2.14)$$

It is shown that equation (2.10) and (2.12) have the same form. One form can be obtained from the other only with the substitution of x_i by y_j and L by L´(or vice versa). The only difference between equation (2.10) and (2.12) is switching points (L and L´). The values of x in (2.10) are indexed in reverse order as in (2.12), which establishes (Fig 2.4)

$$y_1 = x_N,\ y_2 = x_{N-1}, \ldots, y_N = x_1$$

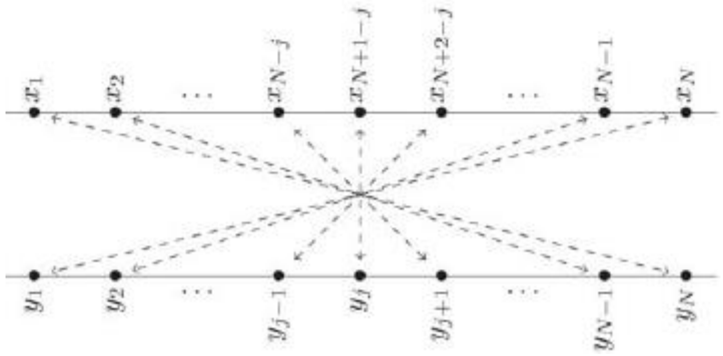


Figure 2.4 Permutation $y_j = x_{N+1-j}$ $(1 \le j \le N)$ that reverses the order in which the values of x are indexed

It is just a permutation (a bijective function) of the N values of x. Equation (2.13) can be thought as an indexation of the N values of x in reverse order.

The computation c_l and c_r can be reduced to one single procedure. It is just necessary to reverse the order, in which the values of x are indexed, and if we are computing c_l then we will need to find a minimum, and if we are computing c_r then we will need to find a maximum. A geometrical interpretation is presented in Fig 2.5(a) and Figure 2.5(b), where each x_i (y_i) is accompanied by its lower $\underline{\mu}_{\tilde{A}}(x_i)$ ($\underline{\mu}_{\tilde{A}}(y_i)$) or upper $\bar{\mu}_{\tilde{A}}(x_i)$ ($\bar{\mu}_{\tilde{A}}(y_i)$) grade of membership.

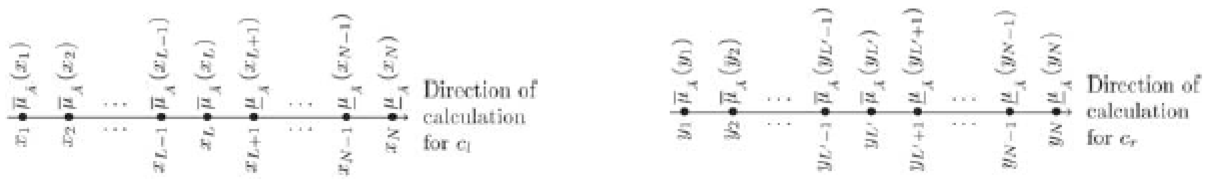

Direction of calculation for c_l
Direction of calculation for c_r

Figure 2.5 (a) Calculation for computing cl by using (2.10). (b) Calculation for computing cr by using (2.12).

If we start form the equation (2.10) by using a similar argument then we will obtain an analogous expression to (2.11). Also we can get the equation (2.15) from equation (2.10)

$$\text{centroid}(A_e(L)) = \frac{\sum_{j=1}^{R'} z_j \overline{\mu}_{\tilde{A}}(z_j) + \sum_{j=R'+1}^{N} z_j \underline{\mu}_{\tilde{A}}(z_j)}{\sum_{j=1}^{R'} \overline{\mu}_{\tilde{A}}(z_j) + \sum_{j=R'+1}^{N} \underline{\mu}_{\tilde{A}}(z_j)} \tag{2.15}$$

here $Z_j = x_{N+1-j}$, $1 \le j \le N$ and $R'=N-L$.

2.4 Description of Karnik–Mendel (KM) Algorithm

There are various methods for type reduction (T2 FS to T1 FS) in IT2 FS, such as Centroid type reduction, Height type reduction, Center-of-sets type reduction etc. Karnik-Mendel algorithm is the basic approach for Centroid type reduction. Before the description of KM algorithm let us assume

$$y(\theta_1, \ldots, \theta_N) \equiv \frac{\sum_{i=1}^{N} x_i \theta_i}{\sum_{i=1}^{N} \theta_i} \tag{2.16}$$

here $(\theta_1, \ldots, \theta_N)$ are the weights of IT2 FS and $\theta_i = \frac{\overline{\mu}_{\tilde{A}}(x_i) + \underline{\mu}_{\tilde{A}}(x_i)}{2}$. Also $\overline{\mu}_{\tilde{A}}(x_i)$ is the upper membership function (UMF) and $\underline{\mu}_{\tilde{A}}(x_i)$ is the lower membership function (LMF) of an IT2 FS.

If the calculus approach is applied for optimizing $y(\theta_{1,\dots,}\theta_N)$ then it should be differentiated with respect to any one of the N value of θ_i, say θ_k, it follows that

$$\frac{\partial y(\theta_{1,\dots,}\theta_N)}{\partial\theta_k} = \frac{\partial}{\partial\theta_k}\left[\frac{\sum_{i=1}^{N} x_i\theta_i}{\sum_{i=1}^{N}\theta_i}\right] = \frac{x_k - y(\theta_{1,\dots,}\theta_N)}{\sum_{i=1}^{N}\theta_i} \tag{2.17}$$

As $\sum_{i=1}^{N}\theta_i > 0$, from equation (2.17) we get,

$$\frac{\partial y(\theta_{1,\dots,}\theta_N)}{\partial\theta_k}\begin{cases}\geq 0 & \text{if } x_k \geq y(\theta_{1,\dots,}\theta_N)\\ < 0 & \text{if } x_k < y(\theta_{1,\dots,}\theta_N)\end{cases} \tag{2.18}$$

Now for maximum and minimum operation this derivative should be zero. But equating $\partial y/\partial\theta_k$ to zero does not give us any information about the value of θ_k that optimizes $y(\theta_1,\dots,\theta_N)$, i.e.,

$$y(\theta_1,\dots,\theta_N) = x_k \Rightarrow \frac{\sum_{i=1}^{N} x_i\theta_i}{\sum_{i=1}^{N}\theta_i} = x_k \Rightarrow \frac{\sum_{i\neq k}^{N} x_i\theta_i}{\sum_{i\neq k}^{N}\theta_i} = x_k \tag{2.19}$$

It is observed that θ_k no longer appears in the final expression in equation (2.19), so that the direct calculus approach does not work.
Equation (2.18) does give the direction in which θ_k should be changed in order to increase or decrease $y(\theta_1,\dots,\theta_N)$:

$$\left.\begin{array}{l}\text{If } x_k > y(\theta_1,\dots,\theta_N) \text{ then } y(\theta_1,\dots,\theta_N) \text{ increases (decreases) as } \theta_k \text{ increases (decreases).}\\ \text{If } x_k < y(\theta_1,\dots,\theta_N) \text{ then } y(\theta_1,\dots,\theta_N) \text{ increases (decreases) as } \theta_k \text{ decreases (increases).}\end{array}\right\} \tag{2.20}$$

The maximum value that θ_k can attain is $\overline{\mu}_{\tilde{A}}(x_k)$ and the minimum value that it can attain is $\underline{\mu}_{\tilde{A}}(x_k)$. Equation (2.20) therefore implies that $y(\theta_1,\dots,\theta_N)$ attains its maximum value, c_r, if

$$\theta_k = \begin{cases}\overline{\mu}_{\tilde{A}}(x_k) & \forall k \ni x_k > y(\theta_1,\dots,\theta_N)\\ \underline{\mu}_{\tilde{A}}(x_k) & \forall k \ni x_k < y(\theta_1,\dots,\theta_N)\end{cases} \tag{2.21}$$

In the same way from the equation (2.18), $y(\theta_1,\ldots,\theta_N)$ attains its minimum value, c_l, if

$$\theta_k = \begin{Bmatrix} \bar{\mu}_{\tilde{A}}(x_k) & \forall k \ni x_k < y(\theta_1,\ldots,\theta_N) \\ \underline{\mu}_{\tilde{A}}(x_k) & \forall k \ni x_k > y(\theta_1,\ldots,\theta_N) \end{Bmatrix} \tag{2.22}$$

There are only two possible choices for θ_k that are stated above, to compute c_r (c_l) θ_k switches only one time between $\bar{\mu}_{\tilde{A}}(x_k)$ and $\underline{\mu}_{\tilde{A}}(x_k)$. A KM algorithm detects the switch point, and, in general, the switch point for c_r, R, is different from the switch point for c_l, L; hence, there are two KM algorithms, one for L switching point and one for R switching point.

The KM algorithms consist of two parts, one for computing y_l in and the other for computing y_r from equation (2.9). Here we assume $y_l \equiv c_l$ and $y_r \equiv c_r$. First we consider the rulebase of an IT2 FLS consisting of N rules assuming the following form:

$\tilde{R}^N$: IF x_1 is $\tilde{X}_1^N$ and · · · and x_I is $\tilde{X}_I^N$, THEN y is Y^N, where X_i^N (i = 1, . . . , I) are IT2 FSs, and $Y^N = \left[\underline{y}^N, \bar{y}^N\right]$ is an interval, which can be understood as the centroid of a consequent IT2 FS, or the simplest Takagi–Sugeno–Kang (TSK) model. In many applications we use $\underline{y}^N = \bar{y}^N$, i.e., each rule consequent is represented by a crisp number.

For an input vector $x' = (x'_1, x'_2, \ldots x'_I)$, typical computations in an IT2 FLS involve the following steps.

1) Compute the membership interval of x'_i on each $X_i^n, \left[\mu_{\underline{X}_i^n}(x'_i), \mu_{\bar{X}_i^n}(x'_i)\right]$, i=1,2,…,I and n=1,2,…,N.

2) Then compute the firing interval of nth rule F^n :

$$F^n = \left[\mu_{\underline{X}_1^n}(x'_1) \times \ldots \times \mu_{\underline{X}_I^n}(x'_I), \mu_{\bar{X}_1^n}(x'_1) \times \ldots \times \mu_{\bar{X}_I^n}(x'_I)\right] \equiv [\underline{\mu}_{\tilde{A}}(x_i), \bar{\mu}_{\tilde{A}}(x_i)] \tag{2.23}$$

where n=1,…, N.

3) Perform Type Reduction (TR) to combine F^n and corresponding rule consequents. There are many such methods. The most commonly used one is the center-of-sets type-reducer.

4)

$$Y_{COS} = \frac{\sum_{n=1}^{N} Y^N F^N}{\sum_{n=1}^{N} F^N} = \bigcup_{\substack{y^n \varepsilon Y^N \\ \mu_{\tilde{A}}(x_n) \varepsilon F^N}}^{n} \frac{\sum_{n=1}^{N} y^n \mu_{\tilde{A}}(x_n)}{\sum_{n=1}^{N} \mu_{\tilde{A}}(x_n)} = [y_l, y_r] \tag{2.24}$$

where $y_l(k) \equiv \min\limits_{k \in [1, N-1]} \dfrac{\sum\limits_{i=1}^{k} \underline{x}_i \overline{\mu}_{\tilde{A}}(x_i) + \sum\limits_{i=k+1}^{N} \underline{x}_i \underline{\mu}_{\tilde{A}}(x_i)}{\sum\limits_{i=1}^{k} \overline{\mu}_{\tilde{A}}(x_i) + \sum\limits_{i=k+1}^{N} \underline{\mu}_{\tilde{A}}(x_i)}$ (2.25)

and $y_r(k) \equiv \min\limits_{k \in [1, N-1]} \dfrac{\sum\limits_{i=1}^{k} \overline{x}_i \underline{\mu}_{\tilde{A}}(x_i) + \sum\limits_{i=k+1}^{N} \overline{x}_i \overline{\mu}_{\tilde{A}}(x_i)}{\sum\limits_{i=1}^{k} \underline{\mu}_{\tilde{A}}(x_i) + \sum\limits_{i=k+1}^{N} \overline{\mu}_{\tilde{A}}(x_i)}$ (2.26)

From equation (2.25) and (2.26) we get the value of k, where k is a possible switching point. In an exhaustive search approach, all k in [1, N − 1] need to be evaluated until the true switch point is identified. Fortunately, y_l and y_r can also be computed more efficiently by the KM algorithms (Table 2.1).

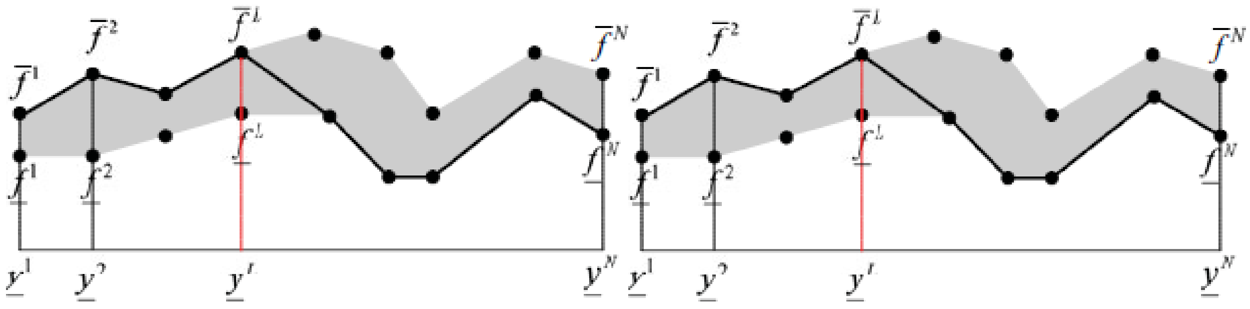
4) After that compute the defuzzified output $y = \dfrac{y_l + y_r}{2}$ for calculating the actual centroid.

The KM algorithm to compute y_l and y_r is given in Table 2.1. The main idea of this algorithm is to find the switching points k (L and R) and defuzzified output i.e., centroid for the type reduction. Take y_l as an example. y_l is the minimum of Y_{cos}. Since y_n increases from the left to the right along the horizontal axis of Fig. 2.6(a), we should choose a large weight (upper bound of the firing interval) for y_n on the left and a small weight (lower bound of the firing interval) for y_n on the right. The KM algorithm for y_l finds the switch point L. For $n \leq L$, the upper bounds of the firing intervals are used to calculate y_l; for $n > L$, the lower bounds are used. This ensures y_l is the minimum. By using same KM algorithm for y_r finds the switching point R (Fig 2.6(b)).

Table 2.1

KARNIK-MENDEL (KM) ALGORITHM

Step	For computing $\mathbf{y_l}$ $\mathbf{y_l} = \min\limits_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \bar{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \le x_2 \ldots \le x_N$	For computing $\mathbf{y_r}$ $\mathbf{y_r} = \max\limits_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \bar{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \le x_2 \ldots \le x_N$
1	Initialize θ_i by setting $\theta_i = \left[\underline{\mu}_{\tilde{A}}(x_i) + \bar{\mu}_{\tilde{A}}(x_i) \right] / 2$, i=1,2,…,N and then compute $y' = y(\theta_1 \ldots \theta_N) = \sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i$	
2	Find k ($1 \le k \le N-1$) such that $x_k \le y' \le x_{k+1}$	
3	Set $\theta_i = \bar{\mu}_{\tilde{A}}(x_i)$ when $i \le k$ and also set $\theta_i = \underline{\mu}_{\tilde{A}}(x_i)$ when $i \ge k+1$ and then compute $y_l(k) \equiv \dfrac{\sum_{i=1}^{k} x_i \bar{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} x_i \underline{\mu}_{\tilde{A}}(x_i)}{\sum_{i=1}^{k} \bar{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} \underline{\mu}_{\tilde{A}}(x_i)}$	Set $\theta_i = \underline{\mu}_{\tilde{A}}(x_i)$ when $i \le k$ and also set $\theta_i = \bar{\mu}_{\tilde{A}}(x_i)$ when $i \ge k+1$ and then compute $y_r(k) \equiv \dfrac{\sum_{i=1}^{k} x_i \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} x_i \bar{\mu}_{\tilde{A}}(x_i)}{\sum_{i=1}^{k} \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} \bar{\mu}_{\tilde{A}}(x_i)}$
4	Check if $y_l(k) = y'$. If yes, stop and set $y_l(k) = y_l$ and set also k to *L*. If no, go to Step5	Check if $y_r(k)=y'$. If yes, stop and set $y_r(k) = y_r$ and set also k to *R*. If no, go to Step5
5	Set $y' = y_l(k)$ and then go to step 2	Set $y' = y_r(k)$ and then go to step 2



(a) (b)

Figure 2.6 Switch points for computing y_l and y_r. (a) Computing y_l : Switch from the upper bounds of the firing intervals to the lower bounds. (b) Computing y_r : Switch from the lower bounds of the firing intervals to the upper bounds.

But there is a high computational cost for iterative KM algorithm. It takes more time to execute the centroid as it is required huge number of iteration. It is more expensive to deploy IT2 FLSs, which may hinder them from certain cost-sensitive real world applications.

There are three categories of methods to reduce their computational cost for finding centroid of an IT2 FS at type-reduction process. The first category consists of five enhancements to the KM algorithms, which are the most popular type-reduction algorithms to date. The second category consists of eleven alternative type-reducers, which have closed-form representations and, hence, are more convenient for analysis. The third category consists of a simplified structure for IT2 FLSs, which can be combined with any algorithms in the first or second category for further computational cost reduction.

2.5 Different types of Enhancements to the Karnik–Mendel Algorithms

Five enhancements to the Karnik-Mendel (KM) algorithm are presented in this section. They are all faster than iterative KM algorithm. Their computational costs may be compared with the original KM algorithms. All of them require that $\{\underline{x}_i^n\}_{n=1,\dots,N}$ and $\{\overline{x}_i^n\}_{n=1,\dots,N}$ are sorted in ascending order, respectively. The Enhancements to the Karnik–Mendel algorithms are

A. Enhanced Karnik–Mendel Algorithm (EKM)

B. Enhanced Karnik–Mendel Algorithm with new initialization (EKMANI)

C. Iterative Algorithm With Stop Condition (IASC)

D. Enhanced Iterative Algorithm With Stop Condition (EIASC)

E. Enhanced Opposite Direction Searching Algorithm (EODS)

2.5.1 Enhanced Karnik-Mendel Algorithm (EKM)

The Enhanced KM (EKM) algorithm [D. Wu and Mendel 2007,2009] are the earliest enhancement to the original KM algorithm. They have three improvements over the KM algorithm. First, a better initialization is used to reduce the number of iterations. Then, the termination condition of the iterations is changed to remove one unnecessary iteration. Finally, a subtle computing technique is used to reduce the computational cost of each iteration. The EKM algorithm is summarized in Table 2.2. The initialization of switching point for calculating y_l is k = (N/2.4), the nearest integer of 2.4 and for calculating y_r is k=(N/1.7), the nearest integer of 1.7. On average, the EKM algorithm can save about two iterations, which corresponds to more than a 39% reduction in computation time. An additional (at least) 23% computational cost can be saved if no sorting of the inputs is needed. The average number of iterations for the EKM algorithm is smaller than that for the original KM algorithm. More interestingly, the average number of iterations for the EKM algorithm is less than one. This is because uniformly and independently distributed $\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)$, and x_i were used in the simulation, and hence, L_0 = [N/2.4] has a good chance to be the final switch point, especially when N is small.

Table 2.2

ENHANCED KARNIK-MENDEL (EKM) ALGORITHM

Step	**For computing y_l** $y_l = \min_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \bar{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$	**For computing y_r** $y_r = \max_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \bar{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$
1	Set k = [N/2.4] (the nearest integer of 2.4) and then compute $a = \sum_{i=1}^{k} x_i \bar{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} x_i \underline{\mu}_{\tilde{A}}(x_i)$ and $b = \sum_{i=1}^{k} \bar{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} \underline{\mu}_{\tilde{A}}(x_i)$ $y' = a/b$;	Set k = [N/1.7] (the nearest integer of 1.7) and then compute $a = \sum_{i=1}^{k} x_i \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} x_i \bar{\mu}_{\tilde{A}}(x_i)$ and $b = \sum_{i=1}^{k} \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} \bar{\mu}_{\tilde{A}}(x_i)$ $y' = a/b$;
2	Find $k' \in [1, N-1]$ such that $x_{k'} \leq y' \leq x_{k'+1}$	
3	Check if $k' = k$ then stop and set $y' = y_l$ and $k = L$. If no go to step 4	Check if $k' = k$ then stop and set $y' = y_r$ and $k = R$. If no go to step 4
4	Compute $s = \text{sign}(k'-k)$ and $a' = a + s \sum_{i=\min(k,k')+1}^{\max(k,k')} x_i \left[\bar{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ $b' = b + s \sum_{i=\min(k,k')+1}^{\max(k,k')} \left[\bar{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ $y''(k') = a'/b'$;	Compute $s = \text{sign}(k'-k)$ and $a' = a - s \sum_{i=\min(k,k')+1}^{\max(k,k')} x_i \left[\bar{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ $b' = b - s \sum_{i=\min(k,k')+1}^{\max(k,k')} \left[\bar{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ $y''(k') = a'/b'$;
5	Set $y' = y''(k')$, $a = a'$, $b = b'$ and $k' = k$, then go to step 2	

2.5.2 Enhanced Karnik-Mendel Algorithm with new initialization (EKMANI)

An Enhanced Karnik-Mendel Algorithm with new initialization (EKMANI) has been proposed by Yeh, Jeng, Lee [2011] to compute the generalized centroid of general T2 FSs. It is based on the fact that two α-planes of an IT2 FS are closed to each other and their resulting centroids are also closed to each other. Therefore, it may be advantageous to use the switch points, which are obtained from the previous α-plane to initialize the switch points in the current α-plane. EKMANI was primarily designed to compute the generalized centroid. It may be also used for type reduction of IT2 FSs. Usually the output of an IT2 FLS changes only a small amount in each step.

The EKMANI algorithm, which is illustrated in Table 2.3, is identical to EKM algorithm for computing y_l. The only difference is that in step (1) if there is a switch point obtained from previous computation, then set l to it; otherwise, set l = [N/2.4].

The EKMANI algorithm is also identical to EKM algorithm for computing y_r. The only difference is that in step (1) if there is a switch point obtained from previous computation, then set r to it; otherwise, set r = [N/1.7].

Table 2.3

ENHANCED KARNIK-MENDEL ALGORITHM WITH NEW INITIALIZATION (EKMANI)

Step	**For computing y_l** $\mathbf{y_l} = \min_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \le x_2 \ldots \le x_N$	**For computing y_r** $\mathbf{y_r} = \max_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \le x_2 \ldots \le x_N$
1	If k is obtained from previous operation then k=L. else set k = [N/2.4] (the nearest integer of 2.4) and then compute $a = \sum_{i=1}^{k} x_i \overline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} x_i \underline{\mu}_{\tilde{A}}(x_i)$ and $b = \sum_{i=1}^{k} \overline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} \underline{\mu}_{\tilde{A}}(x_i)$ y' = a/b;	If k is obtained from previous operation then k=L. else set k = [N/1.7] (the nearest integer of 1.7) and then compute $a = \sum_{i=1}^{k} x_i \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} x_i \overline{\mu}_{\tilde{A}}(x_i)$ and $b = \sum_{i=1}^{k} \underline{\mu}_{\tilde{A}}(x_i) + \sum_{i=k+1}^{N} \overline{\mu}_{\tilde{A}}(x_i)$ y' = a/b;
2	Find k' ∈ [1, N-1] such that $x_{k'} \le y' \le x_{k'+1}$	
3	Check if k' = k then stop and set y' = y_l and k = L. If no go to step 4	Check if k' = k then stop and set y' = y_r and k = R. If no go to step 4
4	Compute s = sign (k'-k) and $a' = a + s \sum_{i=\min(k,k')+1}^{\max(k,k')} x_i \left[\overline{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ $b' = b + s \sum_{i=\min(k,k')+1}^{\max(k,k')} \left[\overline{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ y''(k') = a'/b';	Compute s = sign (k'-k) and $a' = a - s \sum_{i=\min(k,k')+1}^{\max(k,k')} x_i \left[\overline{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ $b' = b - s \sum_{i=\min(k,k')+1}^{\max(k,k')} \left[\overline{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i) \right]$ y''(k') = a'/b';
5	Set y'= y''(k') , a = a', b = b' and k' = k, then go to step 2	

2.5.3 Iterative Algorithm with Stop Condition (IASC)

Melgarejo et al. [2008] proposed an efficient algorithms to compute the generalized centroid of IT2 FSs, which can also be used in TR of IT2 FLSs. This is Iterative Algorithm with Stop Condition (IASC), which is faster than KM, EKM and EKMANI algorithms. IASC is mainly described in Table 2.4. It is based on the fact that $y_l(k)$ in (2.25) first monotonically decreases and then monotonically increases with the increase of 'k', and $y_r(k)$ in (2.26) first monotonically increases and then monotonically decreases with the increase of 'k'. Therefore, the IASC algorithms compute the switch point for y_l from 1 to N – 1 until $y_l(k)$ stops decreasing, at which point y_l is obtained. Similarly, they enumerate the switch point for y_r from 1 to N – 1 until $y_r(k)$ stops increasing, at which point y_r is obtained.

Table 2.4

IASC ALGORITHM

Step	**For computing y_l** $y_l = \min_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$	**For computing y_r** $y_r = \max_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$
1	Initialize $a = \sum_{i=1}^{N} x_i \underline{\mu}_{\tilde{A}}(x_i)$ $b = \sum_{i=1}^{N} \underline{\mu}_{\tilde{A}}(x_i)$ $x_l = \underline{x}^N$ $l = 0$	Initialize $a = \sum_{i=1}^{N} x_i \overline{\mu}_{\tilde{A}}(x_i)$ $b = \sum_{i=1}^{N} \overline{\mu}_{\tilde{A}}(x_i)$ $x_r = \overline{x}^1$ $r = 0$
2	Compute $l = l + 1$ $a = a + \underline{x}^l(\overline{\mu}_{\tilde{A}}(x_l) - \underline{\mu}_{\tilde{A}}(x_l))$ $b = b + (\overline{\mu}_{\tilde{A}}(x_l) - \underline{\mu}_{\tilde{A}}(x_l))$ $c = a/b;$	Compute $r = r + 1$ $a = a - \overline{x}^r(\overline{\mu}_{\tilde{A}}(x_r) - \underline{\mu}_{\tilde{A}}(x_r))$ $b = b - (\overline{\mu}_{\tilde{A}}(x_r) - \underline{\mu}_{\tilde{A}}(x_r))$ $c = a/b;$
3	If $c > x_l$, set $L = l - 1$ and stop; otherwise set $x_l = c$ and go to step 2.	If $c < x_r$, set $R = r - 1$ and stop; otherwise set $x_r = c$ and go to step 2.

2.5.4 Enhanced Iterative Algorithm with Stop Condition (EIASC)

The Enhanced Iterative Algorithm with Stop Condition (EIASC) (Table 2.5) has improved over IASC. The computational cost of EIASC is lower than IASC. This algorithm can also compute the generalized centroid of IT2 FSs, which can also be used in TR of IT2 FLSs. It has two improvements over IASC.

1) The new stopping criterion based on the fact that $y_l(k)$ in (2.25) satisfies

$$y_l(k) \begin{cases} \geq \underline{x}_i, & k \leq L \\ < \underline{x}_i, & k > L \end{cases} \tag{2.27}$$

and $y_r(k)$ in (2.26) satisfies

$$y_r(k) \begin{cases} > \overline{x}_i, & k \leq R \\ \leq \overline{x}_i, & k > R \end{cases} \tag{2.28}$$

Table 2.5

EIASC ALGORITHM

Step	**For computing y_l**	**For computing y_r**
	$\mathbf{y_l} = \min_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i\theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$	$\mathbf{y_r} = \max_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i\theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$
1	Initialize $a = \sum_{i=1}^{N} x_i \underline{\mu}_{\tilde{A}}(x_i)$ $b = \sum_{i=1}^{N} \underline{\mu}_{\tilde{A}}(x_i)$ $L = 0$	Initialize $a = \sum_{i=1}^{N} x_i \overline{\mu}_{\tilde{A}}(x_i)$ $b = \sum_{i=1}^{N} \overline{\mu}_{\tilde{A}}(x_i)$ $R = N$
2	Compute $L = L + 1$ $a = a + \underline{x}^L(\overline{\mu}_{\tilde{A}}(x_L) - \underline{\mu}_{\tilde{A}}(x_L))$ $b = b + (\overline{\mu}_{\tilde{A}}(x_i) - \underline{\mu}_{\tilde{A}}(x_i))$ $y_l = a/b;$	Compute $a = a + \overline{x}^R(\overline{\mu}_{\tilde{A}}(x_R) - \underline{\mu}_{\tilde{A}}(x_R))$ $b = b + (\overline{\mu}_{\tilde{A}}(x_R) - \underline{\mu}_{\tilde{A}}(x_R))$ $y_r = a/b;$ $R = R - 1$
3	If $y_l \leq y^{L+1}$, then stop; otherwise go to step 2.	If $y_r \geq y^R$, then stop; otherwise go to step 2.

2) To compute both y_l and y_r in IASC algorithm, it starts from switching point 1 and increases gradually for identifying the correct switching points. This is reasonable for y_l, since it has been shown in [6] that for a variety of scenarios, its switch point L is smaller than N/2; therefore, it is more proficient to search from L = 1instead of L = N − 1. However, setting the initial switch point R = 1 may not be efficient for y_r, since it has been also shown in [D. Wu and Mendel 2009] that generally its switch point R > N/2. Therefore, it would be more efficient if for y_r one initializes the switch point as R = N − 1 and then gradually decreases it until the correct R is found.

It is clearly shown that based on (2.27) and (2.28) a new initialization of L and R can easily express whether it is on the left or right of the true switching point and hence, the new initialization idea in the EKMANI can also be used in EIASC. It is implemented the corresponding algorithm, where the user can specify an initial L (or R); otherwise, L is initialized to [N/2.4] (R to [N/1.7]), the same as the EKM algorithm. Experiments showed that this new algorithm only outperforms the EIASC when N >1000.

2.5.5 Enhanced Opposite Direction Searching (EODS) Algorithm

Hu et al. [2010, 2012] proposed two algorithms to speed up the KM algorithms. The faster one Enhanced Opposite Direction Searching (EODS) Algorithm is presented in Table 2.6. The EODS algorithms are based on the fact that $\underline{x}^{L} \leq y_l\,(L) \leq \underline{x}^{L+1}$ and $\overline{x}^{R} \leq y_r\,(R) \leq \overline{x}^{R+1}$, where y_l (k) and y_r (k) are defined in (2.25) and (2.26), respectively, and L and R are the switch points for y_l and y_r . For each of y_l and y_r, the EODS algorithm iteratively computes a positive search process and a negative search process. The corresponding switch point is obtained when S_l meets S_r.

The EODS algorithm has two improvements over in [Hu et al. 2012].

1) It is changed S_l and S_r from arrays in [Hu et al.2012] to scalars to reduce the memory requirement, as well as to improve speed.

2) In Step (5), it is simplified the computation by computing y_l and y_r from $\underline{x}^{m}$ and $\overline{x}^{m}$, instead of $\underline{x}^{L}$ and $\overline{x}^{R}$ in [Hu et al. 2012].

The EODS algorithm is the fastest algorithm among the KM algorithms and their five enhanced versions. It reduces the computational cost over all other algorithms. The EODS algorithms seem to be the fastest to use when there are less than 100 rules fired each time, which is usually true in practice. However, they are much more complex than the EIASC algorithms, which are only slightly slower than the EODS algorithms for practical IT2 FLCs. It should be noted that the EODS algorithms only work when N ≥ 3. Therefore, N = 1 and N = 2 need to be considered separately. Fortunately, they rarely occur in practice and there are closed form solutions for y_l and yr for these two special cases.

Table 2.6

EODS ALGORITHM

Step	**For computing y_l** $y_l = \min\limits_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$	**For computing y_r** $y_r = \max\limits_{\forall \theta_i \in [\underline{\mu}_{\tilde{A}}(x_i), \overline{\mu}_{\tilde{A}}(x_i)]} \left(\sum_{i=1}^{N} x_i \theta_i / \sum_{i=1}^{N} \theta_i \right)$ Consider $x_1 \leq x_2 \ldots \leq x_N$
1	If N=1 then $y_l = x_l$ and $l = 1$; elseif N=2 then $y_l = \dfrac{(\underline{x}_1 \overline{\mu}_{\tilde{A}}(x_1)) + (\underline{x}_2 \underline{\mu}_{\tilde{A}}(x_2))}{\overline{\mu}_{\tilde{A}}(x_1) + \underline{\mu}_{\tilde{A}}(x_2)}$ else Initialize $m = 2$ and $n = N-1$ $SLm = (\underline{x}^m - \underline{x}^1)\overline{\mu}_{\tilde{A}}(x_1)$ $SRn = (\underline{x}^N - \underline{x}^n)\underline{\mu}_{\tilde{A}}(x_N)$ $Wls = \underline{\mu}_{\tilde{A}}(x_N)$ $Wrs = \overline{\mu}_{\tilde{A}}(x_1)$	If N=1 then $y_r = x_r$ and $r = 1$; elseif N=2 then $y_r = \dfrac{(\overline{x}_1 \underline{\mu}_{\tilde{A}}(x_1)) + (\overline{x}_2 \overline{\mu}_{\tilde{A}}(x_2))}{\underline{\mu}_{\tilde{A}}(x_1) + \overline{\mu}_{\tilde{A}}(x_2)}$ else Initialize $m = 2$ and $n = N-1$ $SLm = (\overline{x}^m - \overline{x}^1)\underline{\mu}_{\tilde{A}}(x_1)$ $SRn = (\overline{x}^N - \overline{x}^n)\overline{\mu}_{\tilde{A}}(x_N)$ $Wls = \underline{\mu}_{\tilde{A}}(x_1)$ $Wrs = \overline{\mu}_{\tilde{A}}(x_N)$
2	If m = n then go to step 4.	
3	If SLm > SRn then $Wls = Wls + \underline{\mu}_{\tilde{A}}(x_n)$ $n = n - 1$ $SRn = SRn + Wls(\underline{\mu}_{\tilde{A}}(x_{n+1}) - \underline{\mu}_{\tilde{A}}(x_n))$ else $Wrs = Wrs + \overline{\mu}_{\tilde{A}}(x_m)$ $m = m+1$ $SLm = SLm + Wrs(\underline{\mu}_{\tilde{A}}(x_m) - \underline{\mu}_{\tilde{A}}(x_{m-1}))$ Then go to step 2	If SLm > SRn then $Wrs = Wrs + \overline{\mu}_{\tilde{A}}(x_n)$ $n = n - 1$ $SRn = SRn + Wrs(\overline{\mu}_{\tilde{A}}(x_{n+1}) - \overline{\mu}_{\tilde{A}}(x_n))$ else $Wls = Wls + \underline{\mu}_{\tilde{A}}(x_m)$ $m = m+1$ $SLm = SLm + Wls(\overline{\mu}_{\tilde{A}}(x_m) - \overline{\mu}_{\tilde{A}}(x_{m-1}))$ Then go to step 2
4	If $SLm \leq SRn$, then $L = m$ $Wrs = =Wrs + \overline{\mu}_{\tilde{A}}(x_m)$ else $L = m-1$ $Wls = Wls + \underline{\mu}_{\tilde{A}}(x_m)$	If $SLm \leq SRn$, then $R = m$ $Wls = Wls + \underline{\mu}_{\tilde{A}}(x_m)$ else $R = m-1$ $Wrs = Wrs + \overline{\mu}_{\tilde{A}}(x_m)$
5	$y_l = \underline{x}_m + \dfrac{SRn - SLm}{Wrs + Wls}$	$y_r = \overline{x}_m + \dfrac{SRn - SLm}{Wrs + Wls}$

2.6 Different types of Alternative Type-Reduction Algorithms

The iterative KM algorithms have high computational cost as well as their iterative nature makes them difficult to analyze. So it has been proposed many Alternative Type-Reduction algorithms, which have closed-form expressions and are usually faster than KM algorithm. Eleven of them [M. Begian et al. (Ref. no. 13 to 23] are illustrated in this particular section. They are presented in the chronological order. The eleven alternative type-reduction methods are listed below:

A. Gorzalczany Method (G)

B. Liang–Mendel Unnormalized Method (LM)

C. Wu–Mendel Uncertainty Bound Method (UB)

D. Wu–Tan Method (WT)

E. Coupland–John Geometric Method (CJG)

F. Nie–Tan Method (NT)

G. Begian–Melek–Mendel Method (BMM)

H. Greenfield–Chiclana–Coupland–John Collapsing Method (GCCJC)

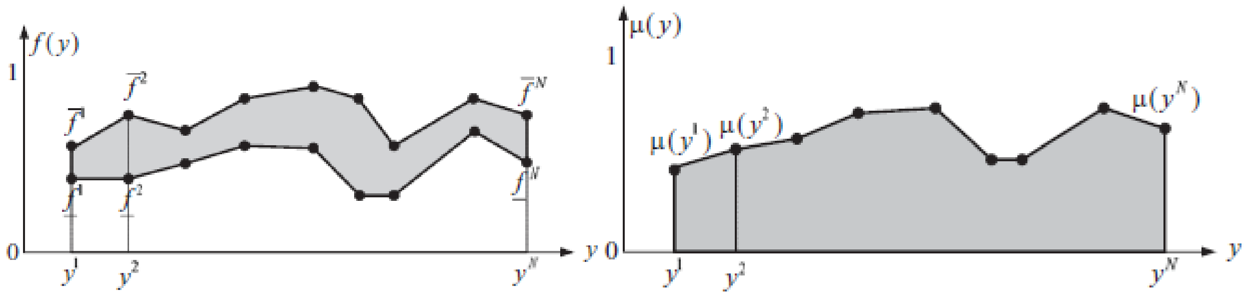
I. Li–Yi–Zhao Method (LYZ)

J. Du–Ying Method (DY)

K. Tao–Taur–Chang–Chang Method (TTCC)

2.6.1 Gorzalczany Method (G)

Gorzalczany proposed two defuzzification methods to obtain a number from the output of the Mamdani inference engine using interval-valued FSs. In his method it is only applied to $\underline{y}^n = \overline{y}^n \equiv y^n$ where n= 1,2,…,N. In this method it is given y^N and the firing intervals of the rules, $\left[\underline{f}^N, \overline{f}^N\right]$, n =1, 2, . . .,N, first we construct a polygon shown in Fig. 11, which can be viewed as a special IT2 FS. At first a polygon is constructed in Fig 2.7(a) which can be viewed as a special IT2 FS.



(a) Polygon used in Gorzalczany's method to compute μ(y) (b) Polygon used in Gorzalczany's method for computing y_G

Figure 2.7

For each point in $\left[y^1, y^N\right]$ it is computed that

$$\mu(y) = \frac{\left(\underline{f} + \overline{f}\right)}{2}.\left[1 - \left(\underline{f} - \overline{f}\right)\right] \tag{2.29}$$

where $\underline{f} - \overline{f}$ represents the bandwidth. Then the defuzzified output can be defined as

$$y_G = \arg\max_y \mu(y) \tag{2.30}$$

Gorzalczany explained that the element y_G adequately fulfills the negotiation between the maximization of the mean value and the minimization of the bandwidth of the inference engine output. The defuzzified output is chosen as the point that divides in half the region under the curve μ(y), i.e., y_G is the solution to the following equation:

$$\int_{y^1}^{y_G} \mu(y)dy = \int_{y_G}^{y^N} \mu(y)dy \tag{2.31}$$

Gorzalczany did not point out how to efficiently compute y_G. However, if we form a granule from μ(y), as shown in Fig. 12, then yG in (2.7(b)) is its centroid. Coupland and John’s method for computing the geometric centroid of an IT2 FS, can be used for this purpose, which can reduce the computational cost.

2.6.2 Liang–Mendel Unnormalized Method (LM)

Liang and Mendel [2000] proposed an unnormalized TR method, in which the defuzzified output is still computed by $y = \frac{y_l + y_r}{2}$, but here y_l and y_r are represented as

$$y_l = \sum_{n=1}^{N} \underline{f}^n y^n \quad \text{and} \quad y_r = \sum_{n=1}^{N} \overline{f}^n y^n$$

where $\underline{y}^n = \overline{y}^n \equiv y^n$ and $\{y^n\}$ do not need to be sorted. This method is called unnormalized because neither y_l nor y_r is normalized by the sum of the firing levels. In the literature and practice, most FLSs use normalized defuzzification.

2.6.3 Wu–Mendel Uncertainty Bound Method (UB)

The uncertainty bound method, proposed by Wu and Mendel [2002], computes the output of the IT2 FLS by $y = \frac{y_l + y_r}{2}$, but here y_l and y_r are also represented as

$$y_l = \sum_{n=1}^{N} \underline{f}^n y^n \quad \text{and} \quad y_r = \sum_{n=1}^{N} \overline{f}^n y^n$$

where $\overline{y}_l = \min\{\underline{y}^{(0)}, \underline{y}^{(N)}\}$ and $\underline{y}_r = \max\{\overline{y}^{(0)}, \overline{y}^{(N)}\}$, we can also write the value of y_l and y_r as

$$\underline{y}_l = \overline{y}_l - \frac{\sum_{n=1}^{N}\left(\overline{f}^n - \underline{f}^n\right)}{\sum_{n=1}^{N}\overline{f}^n \sum_{n=1}^{N}\underline{f}^n} \times \frac{\sum_{n=1}^{N}\underline{f}^n\left(\underline{y}^n - \underline{y}^1\right)\sum_{n=1}^{N}\overline{f}^n\left(\underline{y}^N - \underline{y}^n\right)}{\sum_{n=1}^{N}\underline{f}^n\left(\underline{y}^n - \underline{y}^1\right) + \sum_{n=1}^{N}\overline{f}^n\left(\underline{y}^N - \underline{y}^n\right)} \tag{2.32}$$

and

$$\overline{y}_r = \underline{y}_r + \frac{\sum_{n=1}^{N}\left(\overline{f}^n - \underline{f}^n\right)}{\sum_{n=1}^{N}\overline{f}^n \sum_{n=1}^{N}\underline{f}^n} \times \frac{\sum_{n=1}^{N}\overline{f}^n\left(\overline{y}^n - \overline{y}^1\right)\sum_{n=1}^{N}\underline{f}^n\left(\overline{y}^N - \overline{y}^n\right)}{\sum_{n=1}^{N}\overline{f}^n\left(\overline{y}^n - \overline{y}^1\right) + \sum_{n=1}^{N}\underline{f}^n\left(\overline{y}^N - \overline{y}^n\right)} \tag{2.33}$$

in which $\underline{y}^{(0)} = \dfrac{\sum_{n=1}^{N}\underline{y}^n\underline{f}^n}{\sum_{n=1}^{N}\underline{f}^n}$, $\underline{y}^{(N)} = \dfrac{\sum_{n=1}^{N}\underline{y}^n\overline{f}^n}{\sum_{n=1}^{N}\overline{f}^n}$, $\overline{y}^{(0)} = \dfrac{\sum_{n=1}^{N}\overline{y}^n\underline{f}^n}{\sum_{n=1}^{N}\underline{f}^n}$ and $\overline{y}^{(N)} = \dfrac{\sum_{n=1}^{N}\overline{y}^n\overline{f}^n}{\sum_{n=1}^{N}\overline{f}^n}$

Unlike the KM algorithms, the uncertainty-bound method does not require $\{\underline{y}^n\}$ and $\{\overline{y}^n\}$ to be sorted, although it still needs to identify the minimum and maximum of $\{\underline{y}^n\}$ and $\{\overline{y}^n\}$.

2.6.4 Wu–Tan Method (WT)

Wu and Tan have proposed a closed-form type-reduction and defuzzification method by making use of the equivalent type-1 membership grades. The basic idea of of Wu-Tan method is to first find an equivalent type-1 membership grade $\mu_{X_i^n}(x_i)$ to replace each firing interval $\left[\mu_{\underline{X}_i^n}(x_i), \mu_{\overline{X}_i}(x_i)\right]$, i.e.,

$$\mu_{X_i^n}(x_i) = \mu_{\overline{X}_i^n}(x_i) - h_i^n(x)\left[\mu_{\overline{X}_i^n}(x_i) - \mu_{\underline{X}_i^n}(x_i)\right] \tag{2.34}$$

where $h_i^n(x)$ is a function of the input value x and this function is different for different interval type-2 fuzzy systems (IT2 FSs). Here also x is a function e and $\dot{e}$ and $h_i^n(x) = \alpha e + \beta\dot{e}$, where α and β change with their value of i and n and they were identified by Genetic Algorithms. This property is motivated by the adaptiveness of an IT2 FLC, which means that the embedded T1 FSs used to compute the bounds of the TR interval change as input changes.

Instead of the interval, the firing strengths of the rules become point numbers f^n that are computed from these $\mu_{X_i^n}(x_i)$, the IT2 FLS then becomes an adaptive T1 FLC, and its output is calculated as

$$y = \frac{\sum_{n=1}^{N} y^n f^n}{\sum_{n=1}^{N} f^n}$$

The Wu-Tan (WT) method also does not require $\{y^n\}$ to be sorted.

2.6.5 Coupland–John Geometric Method (CJG)

Coupland and John [14] have proposed a very interesting geometric method for TR and defuzzification of Mamdani IT2 FLSs. This method has extended the TSK IT2 FSs.

At first Coupland and John have constructed a closed polygon (Fig 2.8) from y^n and the corresponding firing intervals. They also relabeled the boundary points as shown in the figure 2.8. In this method there are 2N points on the boundary of the closed polygon, (y^n, f^n) where n=1,2,…,2N, for an IT2 FLS with N rules. Then, the centroid of the polygon is viewed as the defuzzification output (y) of the IT2 FLS:

$$y = \frac{\sum_{n=1}^{2N}\left(y^n + y^{n+1}\right)\left(y^n f^{n+1} - y^{n+1} f^n\right)}{3\sum_{n=1}^{2N}\left(y^n f^{n+1} - y^{n+1} f^n\right)} \tag{2.35}$$

where (y^{2N+1}, f^{2N+1}) is same as (y^1, f^1). It is observed that the geometric method requires $\{y^n\}_{n=1,\ldots,N}$ to be sorted that we can construct a closed polygon.

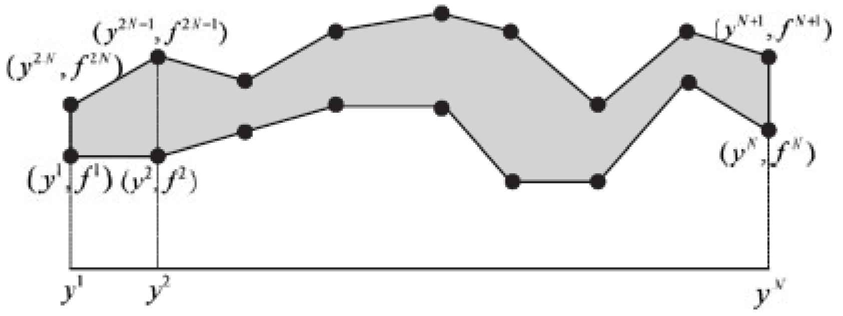


Figure 2.8 A closed polygon representation for Coupland–John method

2.6.6 Nie–Tan Method (NT)

Nie and Tan also enlightened an another closed-form type reduction (TR) and defuzzification method where the centroid of an interval type-2 fuzzy logic systems (IT2 FSs) may be computed as

$$y = \frac{\sum_{n=1}^{N} y^n(\underline{f}^n + \overline{f}^n)}{\sum_{n=1}^{N} \underline{f}^n + \overline{f}^n} \tag{2.36}$$

In case of Nie–Tan method does not require $\{y^n\}_{n=1,\ldots,N}$ to be sorted. This method is the special case of Wu-Tan (WT) method when the value of $h_i^n(x) = 0.5$. However if $h_i^n(x)$ is constant for all of inputs then the resulting IT2 FLS loses adaptiveness. This is one of the fundamental differences between IT2 FS and IT2 FLCs.

2.6.7 Begian–Melek–Mendel Method (BMM)

Begian et al. [2008] proposed another closed-form TR and defuzzification method for IT2 FLSs for calculating the centroid, i.e.,

$$y = \alpha \frac{\sum_{n=1}^{N} \underline{f}^{n} y^{n}}{\sum_{n=1}^{N} \underline{f}^{n}} + \beta \frac{\sum_{n=1}^{N} \overline{f}^{n} y^{n}}{\sum_{n=1}^{N} \overline{f}^{n}} \tag{2.37}$$

where α and β are the adjustable coefficients. Here it is observed that the output of an IT2 FLS is a combination of the outputs of two T1 FLSs. One is constructed only from the LMFs and the other is constructed only from the UMFs of an IT2 FS.

The Begian–Melek–Mendel (BMM) method does not require $\{y^n\}$ to be sorted. It is also similar to Niewiadomski et al.'s [39] fourth method for TR of IT2 FSs. However, Niewiadomski et al. have not extended their methods to IT2 FLSs.

The BMM method requires $\underline{y}^n = \overline{y}^n \equiv y^n$. But Li et al. [2011] has extended it to the case that $\underline{y}^n \neq \overline{y}^n$, i.e.,

$$y = \alpha \frac{\sum_{n=1}^{N} \underline{f}^{n} \underline{y}^{n}}{\sum_{n=1}^{N} \underline{f}^{n}} + \beta \frac{\sum_{n=1}^{N} \overline{f}^{n} \overline{y}^{n}}{\sum_{n=1}^{N} \overline{f}^{n}} \tag{2.38}$$

It is experimented that equation (2.37) and equation (2.38) have same computational cost. Therefore it is only considered the BMM method. It should be notified that Castillo et al. [25] also approached a different method to find out the centroid of an IT2 Fs, which is similar to the BMM method.

2.6.8 Greenfield–Chiclana–Coupland–John Collapsing Method (GCCJC)

Greenfield et al. [2008] proposed a collapsing method for TR of IT2 FLSs, where each of the IT2 FS is replaced by a representative embedded T1 FS whose membership grades are computed recursively. The representative embedded T1 FS can be approximated by a pseudo representative embedded T1 FS to simplify the computation. The membership function in GCCJC method is represented as

$$\mu_X(x) = \frac{\mu_{\underline{X}}(x) + \mu_{\overline{X}}(x)}{2} \tag{2.39}$$

The IT2 FLS is reduced to a T1 FLS if once all IT2 FSs in an IT2 FLS are replaced by their pseudo representative embedded T1 FSs. The defuzzification method is straightforward here. In this collapsing method does not require $\{y^n\}$ to be sorted.

The GCCJC method is almost identical to NT method. In fact, when there is only one input, these two methods are identical. But when there are more than one input, then these two methods are different, because the firing level of $\tilde{R}^n$ in the Greenfield–Chiclana–Coupland–John–Collapsing(GCCJ) method is

$$f^n_{GCCJC} = \prod_{i=1}^{I} \frac{\mu_{\underline{X}_i^n}(x_i) + \mu_{\underline{X}_i^n}(x_i)}{2} \tag{2.40}$$

whereas in NT method the firing level of $\tilde{R}^n$ is defined as

$$f^n_{NT} = \frac{\prod_{i=1}^{I} \mu_{\underline{X}_i^n}(x_i) + \prod_{i=1}^{I} \mu_{\overline{X}_i^n}(x_i)}{2} \tag{2.41}$$

Greenfield also proposed another method for TR, which is called the sampling method, where only a relatively small random sample of the totality of embedded T1 FSs is processed. But the output of this particular method is not deterministic. So this method is not considered here.

2.6.9 Li–Yi–Zhao Method (LYZ)

Li et al. [2008] proposed a new TR method based on interval analysis without considering the dependence of f^n in the numerator and denominator of (2.24). They still computed the output of the IT2 FLS by $y = \frac{y_l + y_r}{2}$, where y_l and y_r are defined as,

$$y_l = \min\left[\frac{\sum_{n=1}^{N} \min\left(\underline{f}^n \underline{y}^n, \overline{f}^n \underline{y}^n\right)}{\sum_{n=1}^{N} \underline{f}^n}, \frac{\sum_{n=1}^{N} \min\left(\underline{f}^n \underline{y}^n, \overline{f}^n \underline{y}^n\right)}{\sum_{n=1}^{N} \overline{f}^n}\right] \tag{2.42}$$

$$y_r = \max\left[\frac{\sum_{n=1}^{N} \max\left(\underline{f}^n \overline{y}^n, \overline{f}^n \overline{y}^n\right)}{\sum_{n=1}^{N} \underline{f}^n}, \frac{\sum_{n=1}^{N} \max\left(\underline{f}^n \overline{y}^n, \overline{f}^n \overline{y}^n\right)}{\sum_{n=1}^{N} \overline{f}^n}\right] \tag{2.43}$$

In this LYZ method $\{y^n\}$ need not to be sorted.

Li et al. [2008] also showed that the $[y_l, y_r]$ which are computed from the KM algorithms is a subset of the $[y_l, y_r]$ computed earlier, and the absolute value of the difference between the defuzzified output computed by the KM algorithms and their defuzzified output is upper bounded by

$$\max\left(\max_n \left|\underline{y}^n\right|, \max_n \left|\overline{y}^n\right|\right) \cdot \frac{\sum_{n=1}^{N}\left(\overline{f}_n - \underline{f}^n\right)}{\sum_{n=1}^{N} \underline{f}^n} \tag{2.44}$$

But this is very loose bound, especially when $\sum_{n=1}^{N}\underline{f}^n$ approaches zero. This should be very clear by giving a simple example. We considered that only two rules are fired, i.e., $\left[\underline{f}^1,\overline{f}^1\right]$ =[0.001, 0.2], $\left[\underline{y}^1,\overline{y}^1\right]$ = [0.5, 0.6], $\left[\underline{f}^2,\overline{f}^2\right]$ =[0,0.8], $\left[\underline{y}^2,\overline{y}^2\right]$ =[0.9,1]. The bounds are computed to be 999, which is 999 times of the maximum consequent $\overline{y}^2$. In fact, [y_l, y_r] are computed by the KM algorithms is [0.5, 0.9995], whereas [y_l, y_r] are computed by the LYZ method is [0.0005, 920]. Clearly, it is a huge difference.

2.6.10 Du–Ying Method (DY)

Du and Ying [2010] have proposed of an average defuzzifier of an IT2 FS. By all possible combinations of lower and upper firing intervals, at first 2^N crisp outputs are computed. So it can be written that

$$y_m = \frac{\sum_{n=1}^{N} y^n f^{n^*}}{\sum_{n=1}^{N} f^{n*}}, \text{ m=1,2,…,}2^N,$$

where $f^{n^*} \in \left\{\underline{f}^n, \overline{f}^n\right\}$. Then the final defuzzified output (y) can be computed as the average of all 2^N crisp outputs (y_m). We can write the defuzzified output as

$$y = \frac{1}{2^N}\sum_{m=1}^{2^N} y_m \tag{2.45}$$

The DY method also does not require {y^n} to be sorted. Also the DY method makes the analysis of an IT2 FS easier but it has a high computational cost than iterative KM algorithm. Its computational cost increases exponentially if the numbers of rules are increased, since there are 2^N of crisp output of T1 FLSSs.

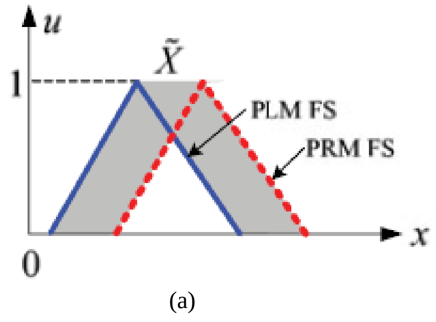
2.6.11 Tao–Taur–Chang–Chang Method (TTCC)

Tao et al. [2012] proposed a simplified IT2 FLS, whose defuzzified output is computed as

$$y = \alpha y_{PLM} + (1-\alpha) y_{PRM} \tag{2.46}$$

where y_{PLM} is the output of a T1 FLS constructed only from the possible-left-most (PLM) embedded T1Fss, and y_{PRM} is the output of a T1 FLS constructed only from the possible-right-most (PRM) embedded T1FSs. An IT2 FS is computed by blurring a triangular T1 FS left and right, and hence, the possible-leftmost and possible-right-most embedded T1 FSs can be easily identified, as shown in Fig. 2.9(a). However Tao et al. did not discuss how to identify these embedded T1 FSs for IT2 FSs with arbitrary FOUs. Possible-left-most and possible-right-most embedded T1 FSs for an arbitrary FOU is constructed in Fig 2.9(b). TTCC is the special case which is represented here.

The Tao–Taur–Chang–Chang (TTCC) method does not require $\{y^n\}$ to be sorted. TTCC method is same as BMM method. In this both methods the output of an It2 FLS is computed as a linear combination of the outputs of two T1 FLSs, which are constructed from the embedded T1 FSs. However the BMM method uses the upper and lower MFs, whereas the TTCC method uses the possible-left-most and possible-rightmost convex and normal embedded T1 FSs.



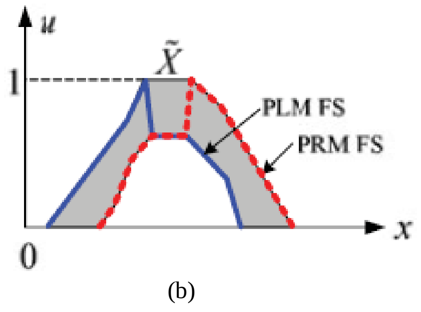


Figure 2.9 (a) PLM and PRM embedded T1 FSs (b) PLM and PRM embedded T1 FSs for an arbitrary FOU.

2.7 Simplified Interval Type-2 Fuzzy Logic Systems & its explanation

The third category is the simplified structure of IT2 fuzzy logic systems (FLSs).It can be merged with any algorithms in the first or second category to further decrease the computational cost.

A unique observation can be made from the IT2 FLC. Its ability to eliminate oscillations can be attributed to its control surface near the steady state. To save computational cost, while maintaining their superior ability to eliminate oscillations, it is proposed a simplified architecture for IT2 FLCs where IT2 FSs are only used for the most critical regions in the input domains. The rest of the input domains are covered by T1 FSs.

Now a PI controller is considered. The control surface near e = 0 and $\dot{e}$ = 0 is mainly responsible for eliminating the oscillations. A simplified architecture is shown in figure 2.10. Here IT2 FS is used to cover the areas around e = 0 and $\dot{e}$ = 0 and T1 FSs for the rest of the input domains.

A simplified IT2 FLC using trapezoidal MFs has two parts—a T1 part and an IT2 part. When the state of the plant is in different operating regions then the different fuzzy partitions will be activated. The FLC behaves like a T1 FLC during the transient stage, since no IT2 FSs are fired. When the output approaches the set point, IT2 FSs will be fired and the plant is controlled by an IT2 FLC. Smoother control signals will be created, which help remove oscillations.

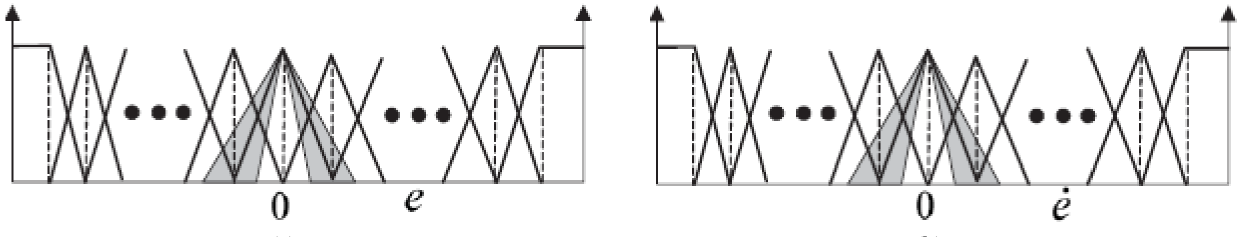

(a) (b)

Figure 2.10 MFs of the simplified IT2 FLS in (a) the *e* domain, and (b) the $\dot{e}$ domain. (Note that the middle MF in each domain is an IT2 FS. All other MFs are T1 FSs.)

There are two approaches to design a simplified IT2 FLC:

1) *The one-step approach*, where we specify the number of IT2 FSs near the steady state and then design the simplified IT2 FLC through experience or optimization algorithms;

2) *The two-step approaches*, where we design a baseline T1 FLC first, change some T1 FSs near the steady state to be IT2 FSs, and retune the parameters using an optimization algorithm.

A simplified IT2 FLC, which used one IT2 FS around e = 0 and one around $\dot{e}$ = 0, outperformed a T1 FLC with the same number of MFs and showed similar performance as an IT2 FLC whose MFs are all IT2 FSs. A simplified IT2 FLC, which used only one IT2 FS in the $\dot{e}$ domain, outperformed a T1 FLC with the same number of MFs and showed similar performance as a T1 FLC with more MFs and an IT2 FLC whose all MFs are IT2 FSs.

There are two categories of methods for TR and defuzzification of the simplified IT2 FLC. The first category is to use the KM algorithms or their enhanced versions and the second category is to use the alternative TR algorithms. The second category of methods is very straightforward. Therefore, only the first category of methods is described in this section.

A simplified IT2 FLC is considered where M out of the N rules contain only T1 FS in the antecedent. The remaining N –M rules have at least one IT2 FS in the antecedent. Hence there should be M crisp firing strengths, f^n, n=1,2,…,M and N-M interval firing strengths, F^n, n=M+1, M+2,…, N. In this particular case the type-reducer in equation 2.24 becomes

$$Y_{cos} = \frac{\sum_{n=1}^{M} Y^n f^n + \sum_{n=M+1}^{N} Y^n F^n}{\sum_{n=1}^{M} f^n + \sum_{n=M+1}^{N} F^n}$$

$$= \frac{\beta + \sum_{n=M+1}^{N} Y^n F^n}{\alpha + \sum_{n=M+1}^{N} F^n}$$

$$= \frac{\beta}{\alpha} + \frac{\sum_{n=M+1}^{N} Y^n F^n - \frac{\beta}{\alpha}\sum_{n=M+1}^{N} F^n}{\alpha + \sum_{n=M+1}^{N} F^n}$$

$$= \frac{\beta}{\alpha} + \frac{\sum_{n=M+1}^{N}\left(Y^n - \frac{\beta}{\alpha}\right) F^n}{\alpha + \sum_{n=M+1}^{N} F^n} \tag{2.47}$$

where $$\alpha = \sum_{n=1}^{M} f^n$$

$$\beta = \sum_{n=1}^{M} Y^n f^n = \left[\sum_{n=1}^{M} \underline{y}^n f^n, \sum_{n=1}^{M} \overline{y}^n f^n\right]$$

$$\frac{\beta}{\alpha} = \left[\frac{\sum_{n=1}^{M} \underline{y}^n f^n}{\sum_{n=1}^{M} f^n}, \frac{\sum_{n=1}^{M} \overline{y}^n f^n}{\sum_{n=1}^{M} f^n}\right]$$

$$Y^n - \frac{\beta}{\alpha} = \left[\underline{y}^n - \frac{\sum_{n=1}^{M} \underline{y}^n f^n}{\sum_{n=1}^{M} f^n}, \overline{y}^n - \frac{\sum_{n=1}^{M} \overline{y}^n f^n}{\sum_{n=1}^{M} f^n}\right]$$

It is also defined Y_*^n and F^{N+1} as $$Y_*^n = \begin{cases} Y^n - \frac{\beta}{\alpha}, & n = M+1, M+2, \ldots, N \\ 0, & n = M+1 \end{cases} \tag{2.48}$$

and $F^{N+1} = \alpha$ (2.49)

and equation (2.47) can be further simplified as

$$Y_{cos} = \frac{\beta}{\alpha} + \frac{\sum_{n=M+1}^{N+1} Y_*^n F^n}{\sum_{n=M+1}^{N+1} F^n} \tag{2.50}$$

The second term on the right-hand side of (2.50) can be calculated by the KM algorithms or their enhanced versions, which are given in section 2.6.

A simplified IT2 FLC can save a significant amount of computational cost over a full IT2 FLC, especially when the number of rules is large and the EODS algorithms or the WT (NT) method is used in TR. Particularly, the simplified IT2 FLC using the WT (NT) method has the fastest speed.

2.8 Conclusion

In this chapter we have made a comprehensive overview of the three categories of IT2 FLSs. The first category consists of five enhancements to the KM algorithms. The second category consists of eleven alternative type-reducers, which have closed-form representation and, hence, are more suitable for analysis. However their computational cost may obstruct them from using in certain cost sensitive real world application. So we introduced a third category which can be combined with any algorithm of first or second category and is known as a simplified structure for IT2 FLCs. But the first category cannot be combined with the second category and vice-versa. However, it is important to note that the first two categories of methods can be applied to all IT2 FLSs, whereas the simplified structure is only designed for control applications.

References

[1] J. M. Mendel and F. Liu, “Super-exponential convergence of the Karnik-Mendel algorithms for computing the centroid of an interval type-2 fuzzy set,” IEEE Transactions on Fuzzy Systems, vol. 15, no. 2, pp. 309–320, April 2007.

[2] J. M. Mendel and H. Wu, “New results about the centroid of a n interval type-2 fuzzy set, including the centroid of a fuzzy granule,” Information Sciences, vol. 177, pp. 360–377, 2007.

[3] J. M. Mendel, “On centroid calculations for type-2 fuzzy sets,” Appl. Comput. Math., vol. 10, no. 1, pp. 88–96, 2011.

[4] O. Salazar, J. Soriano, and H. Serrano, “A short note on t he centroid of an interval type-2 fuzzy set,” in Proceedings of Workshop on Engineering Applications (WEA), Bogotá, Colombia, May 2011.

[5] D. Wu and J. M. Mendel, “Enhanced Karnik–Mendel algorithms for interval type-2 fuzzy sets and systems,” in Proc. North Amer. Fuzzy Inf. Process. Soc., San Diego, CA, Jun. 2007, pp. 184–189.

[6] D. Wu and J. M. Mendel, “Enhanced Karnik–Mendel algorithms,” IEEE Trans. Fuzzy Syst., vol. 17, no. 4, pp. 923–934, Aug. 2009.

[7] C.-Y. Yeh, W.-H. Jeng, and S.-J. Lee, “An enhanced type-reduction algorithm for type-2 fuzzy sets,” IEEE Trans. Fuzzy Syst., vol. 19, no. 2, pp. 227–240, Apr. 2011.

[8] K. Duran, H. Bernal, and M. Melgarejo, “Improved iterative algorithm for computing the generalized centroid of an interval type-2 fuzzy set,” in Proc. North Amer. Fuzzy Inf. Process. Soc., New York, May 2008, pp. 1–5.

[9] M. Melgarejo, “A fast recursive method to compute the generalized centroid of an interval type-2 fuzzy set,” in Proc. North Amer. Fuzzy Inf. Process. Soc., San Diego, CA, Jun. 2007, pp. 190–194.

[10] D. Wu and M. Nie, “Comparison and practical implementation of type reduction algorithms for type 2 fuzzy sets and systems,” in Proc. IEEE Int. Conf. Fuzzy Syst., Taipei, Taiwan, Jun. 2011, pp. 2131–2138.

[11] H. Z. Hu,G. Zhao, and H.N.Yang, “Fast algorithm to calculate generalized centroid of interval type-2 fuzzy set,” Control Decis., vol. 25, no. 4, pp. 637–640, 2010.

[12] H. Hu, Y.Wang, and Y. Cai, “Advantages of the enhanced opposite direction searching algorithm for computing the centroid of an interval type-2 fuzzy set,” Asian J. Control, vol. 14, no. 6, pp. 1–9, 2012.

[13] M. Begian, W. Melek, and J. Mendel, “Stability analysis of type-2 fuzzy systems,” in Proc. IEEE Int. Conf. Fuzzy Syst., Jun. 2008, pp. 947–953.

[14] S. Coupland and R. I. John, “Geometric type-1 and type-2 fuzzy logic systems,” IEEE Trans. Fuzzy Syst., vol. 15, no. 1, pp. 3–15, Feb. 2007.

[15] X. Du and H. Ying, “Derivation and analysis of the analytical structures of the interval type-2 fuzzy-PI and PD controllers,” IEEE Trans. Fuzzy Syst., vol. 18, no. 4, pp. 802–814, Aug. 2010.

[16] M. Gorzalczany, “Decision making in signal transmission problems with interval-valued fuzzy sets,” Fuzzy Sets Syst., vol. 23, pp. 191–203, 1987.

[17] S. Greenfield, F. Chiclana, S. Coupland, and R. John, “The collapsing method of defuzzification for discretised interval type-2 fuzzy sets,” Inf. Sci., vol. 179, no. 13, pp. 2055–2069, 2008.

[18] C. Li, J. Yi, and D. Zhao, “A novel type-reduction method for interval type-2 fuzzy logic systems,” in Proc. 5th Int. Conf. Fuzzy Syst. Knowl. Discov., vol. 1, Jinan, China, Mar. 2008, pp. 157–161.

[19] Q. Liang and J. M. Mendel, “Equalization of nonlinear time-varying channels using type-2 fuzzy adaptive filters,” IEEE Trans. Fuzzy Syst., vol. 8, no. 5, pp. 551–563, Oct. 2000.

[20] M. Nie and W. W. Tan, “Towards an efficient type-reduction method for interval type-2 fuzzy logic systems,” in Proc. IEEE Int. Conf. Fuzzy Syst., Jun. 2008, pp. 1425–1432.

[21] C. W. Tao, J. S. Taur, C.-W. Chang, and Y.-H. Chang, “Simplified type-2 fuzzy sliding controller for wing rock system,” Fuzzy Sets Syst., 2012, to be published.

[22] D.Wu and W.W. Tan, “Computationally efficient type-reduction strategies for a type-2 fuzzy logic controller,” in Proc. IEEE Int. Conf. Fuzzy Syst., Reno, NV, May 2005, pp. 353–358.

[23] H.Wu and J. M. Mendel, “Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems,” IEEE Trans. Fuzzy Syst., vol. 10, no. 5, pp. 622–639, Oct. 2002.

[24] A. Niewiadomski, J. Ochelska, and P. Szczepaniak, “Interval-valued linguistic summaries of databases,” Control Cybern., vol. 35, no. 2, pp. 415– 443, 2006.

[25] O. Castillo, P. Melin, A. Alanis, O. Montiel, and R. Sepulveda, “Optimization of interval type-2 fuzzy logic controllers using evolutionary algorithms,” Soft Comput., vol. 15, no. 6, pp. 1145–1160, Jun. 2011.

Chapter 3

Matlab Simulation and Comparison

This chapter provides a comprehensive overview and comparison of three categories of methods to reduce their computational cost. All the methods are compared in control surface and evolutionary IT2 FLCs using both Gaussian MFs and Trapezoidal MFS. Calculation of Centroid, Fuzziness, Cardinality, Variance and Skewness using five enhanced to KM algorithm are presented in the chapter. This chapter also illustrates the statistics of the absolute difference between the outputs of alternative TR algorithms and those of the KM algorithms.

3.1 Introduction

In the previous chapter we have discussed about the KM algorithm for calculating the centroid of type-reduction method in IT2 FS. It is also discussed about the five enhancement versions to the KM algorithm and eleven alternative type reduction methods to minimize the computational cost. Simplified Interval Type-2 Fuzzy Logic Systems is another method, which can be combined with any algorithms in the first or second category for further computational cost reduction. Here we represent the comparison of different methods using fuzzy logic controller (FLC) and control surface. It can be used both Gaussian and Trapezoidal membership functions for these comparisons.

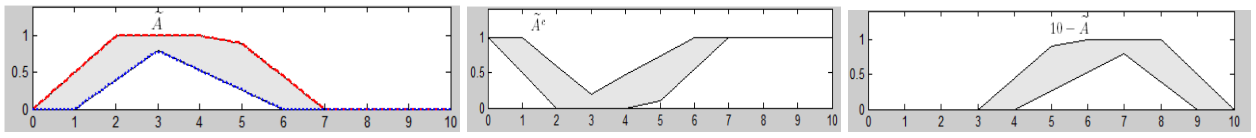
All the experiments are presented next to compare the computational cost of all algorithms under the platform of assembly desktop computer with Intel (R) Pentium (R) 4 CPU @ 2.8 GHz and 2-GB memory, running Windows XP Home Premium 32-bit and MATLAB R2009a.

In all the experiments, we compute the control surfaces of IT2 FLCs using Gaussian MFs and Trapezoidal MFs. This demonstrates the overall computational cost of different algorithms because every input in the input domain is considered and all inputs have the same weight in performance evaluation. In the second experiment, we compare the performance of the different algorithms in IT2 FLC design using evolutionary algorithms, in which a step response is used to evaluate the IT2 FLCs. This demonstrates the practical computational cost of different algorithms because, in practice, different regions of the control surface have different firing frequencies, e.g., the center portion of the control surface is generally fired more often than the boundaries. The step response enables us to simulate this behavior.

From all the five enhancements version of KM algorithm, we have also computed the centroid, fuzziness, cardinality, variance and skewness for a given example.

3.2 Comparison of the Defuzzified Outputs of five Enhanced to KM algorithms

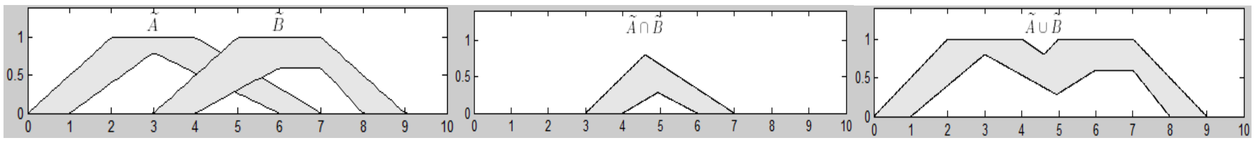
At first we consider an IT2 FS Ã whose upper membership function (UMF) and lower membership function (LMF) is presented here in Fig 3.1(a). There may be different numbers of samples for UMF and LMF, e.g. xUMF=[0 2 4 5 7]; uUMF=[0 1 1 .9 0]; xLMF=[1 3 6]; uLMF=[0 .8 0]; The complement of this IT2 FS, $\tilde{A}^c$ is presented in Fig 3.1(b). The antonym is also shown in Fig 3.1(c).



(a) An IT2 FS Ã (b) Complement of Ã (c) Antonym of Ã

Figure 3.1 Complement and Antonym of an IT2 FS Ã

Also the value of two fuzzy sets A and B are given that A=[0 2 4 7 1 3 3 6 0.8], and B=[3 5 7 9 4 6 7 8 0.6] in Fig 3.2 (a). The intersection and union of these two fuzzy sets are given Fig 3.2 (b) and 3.2 (c) respectively.



(a) Two fuzzy sets A and B (b) Intersection of A and B (c) Union of A and B

Figure 3.2 Intersection and Union of two fuzzy sets A and B

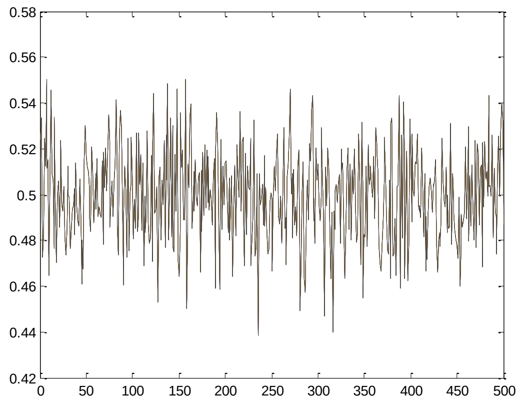
The centroid of all five enhancements to KM algorithms including KM methods are computed and their fuzziness, cardinality, variance, skewness are illustrated in Table 3.1. From the table we get the same centroids for KM, EKM an EKMANI methods. The calculating centroid of IASC method is close to the KM methods but the calculating centroid of EAISC is not so close to the KM methods. The fuzziness and cardinality of all these enhancements are same for this experiment. The variance and skewness of KM, EKM and EKMANI algorithms are equal. The variance of EIASC algorithm is high than all other methods. But in EODS method the variance is identical to the KM method. Now the skewness of KM, EKM, EKMANI and EODS methods are same. The skewness of IASC method is closed as the above mentioned four methods. But in case of EIASC method the skewness is very large among these enhanced KM methods.

Table 3.1 Comparison of outputs of five enhanced to KM algorithms

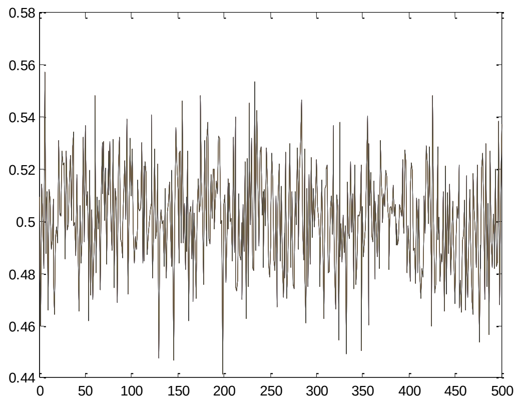
Enhancement to the KM methods	Centroid	Fuzziness	Cardinality	Variance	Skewness
KM	3.3156	0.3924	0.4596	1.8034	0.4345
EKM	3.3156	0.3924	0.4596	1.8034	0.4345
EKMANI	3.3156	0.3924	0.4596	1.8034	0.4345
IASC	3.3137	0.3924	0.4596	1.8035	0.4463
EIASC	3.1650	0.3924	0.4596	1.8324	1.3821
EODS	3.3156	0.3924	0.4596	1.8034	0.4345

3.3 Performance comparison of the five Enhanced to KM algorithms using Gaussian MFs and Trapezoidal MFs

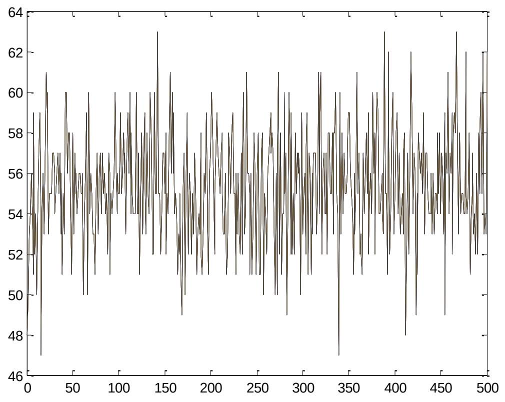
Now it is compared the performance of all the five enhanced algorithms including KM algorithm. Here is seen that 500 rules (N) are fired and the overall results are shown in Fig 3.3 (a) and (b) using Gaussian MFs and Trapezoidal MFs respectively. The performance of all these methods using both membership functions are varied in between 0.44 and 0.55. These performances depend on the number of rules that are fired. The left centroid y_l of all these methods using two types of membership functions are illustrated in Fig 3.3 (c) and (d) respectively. They also depend on the total number of firing rules. All the performance using Trapezoidal MFs are better than Gaussian MFs. It is also seen in Fig 3.3(e) and (f) that for calculating the right centroid y_r using Gaussian MFs has better performance than Trapezoidal MFs.



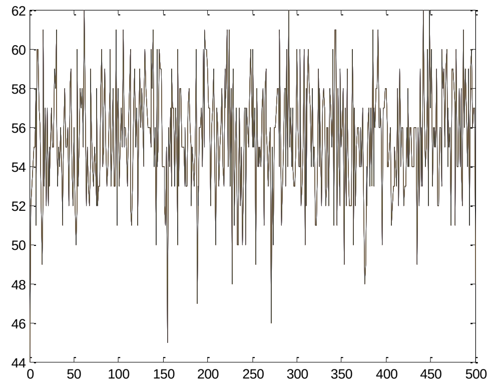
(a) Performance of all methods using Gaussian MFs



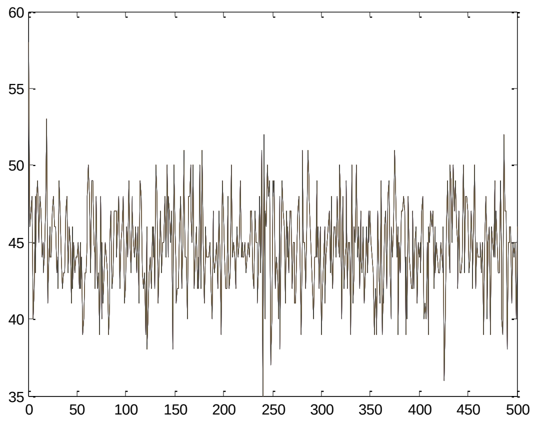
(b) Performance of all methods using Trapezoidal MFs



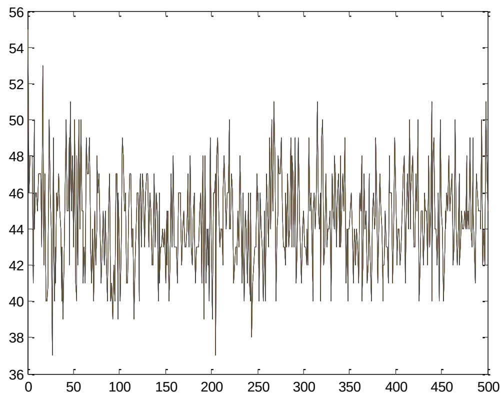
(c) Performance of left centroids using Gaussian MFs



(d) Performance of left centroids using Trapezoidal MFs



(e) Performance of right centroids using Gaussian MFs



(f) Performance of right centroids using Trapezoidal MFs

Figure 3.3 Performance comparison of five enhanced to KM algorithms

3.4 Computational cost comparison of five enhanced KM algorithms

3.4.1 In control surface using Gaussian MFs

First, two-input single-output IT2 PI FLCs using Gaussian MFs are considered. Each input domain consisted of M MFs, and M = {2, 3, . . . , 10, 20, . . . , 50} were used. The center of each MF was generated as a uniformly distributed random number in [−1, 1] using MATLAB command 2*rand-1. The uncertain standard deviations of each MF were generated as uniformly distributed random numbers in [0.1, 0.5] using MATLAB command 0.1*sort(rand(1,2))+0.4. The crisp consequent of each rule was generated as a uniformly distributed random number in [−2, 2] using MATLAB command 4*rand-2. Each input domain was discretized into ten points, and hence computing a complete control surface requires 10 × 10 = 100 TRs. An example of the generated MFs (M = 3) and the corresponding control surface are shown in Fig. 3.4(a) and Fig 3.4(b) respectively.

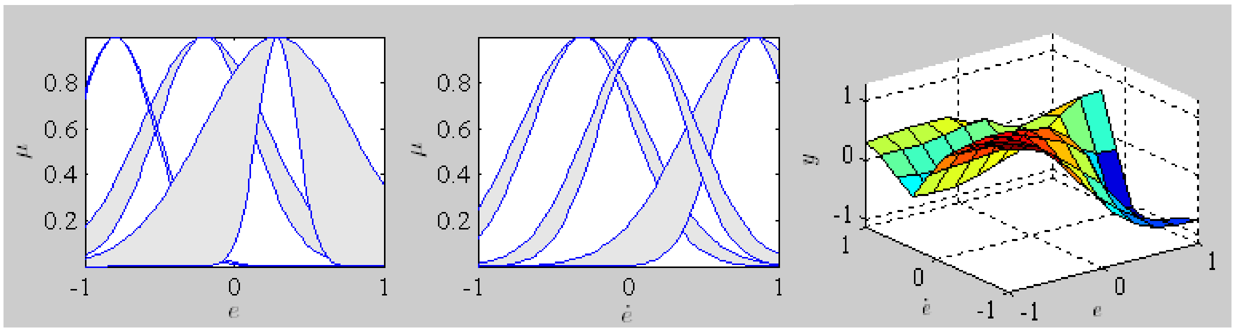


Figure 3.4 (a) A generated Gaussian MFs (b) The corresponding control surface

To make the results statistically meaningful, it is generated 100 random IT2 FLCs for each M and recorded the time that the five enhanced to the KM algorithms were used to perform these 100×100=10 000 TRs. To compare the computational cost of IT2 FLCs with T1 FLCs, we also recorded the computation time for baseline T1 FLCs, whose MFs were the UMFs of the corresponding IT2 FSs. The results are shown in Fig. 5 and the Table 3.2. Note that for fair comparison with the alternative TR methods in the next section, in Fig. 3.5 and Table 3.2, we include the time for computing the firing intervals of the rules,4 because some alternative TR algorithms may compute crisp firing strengths instead of firing intervals. Observe from Fig. 3.5 the following points.

1. All the five enhanced algorithms are always faster than the KM algorithms.

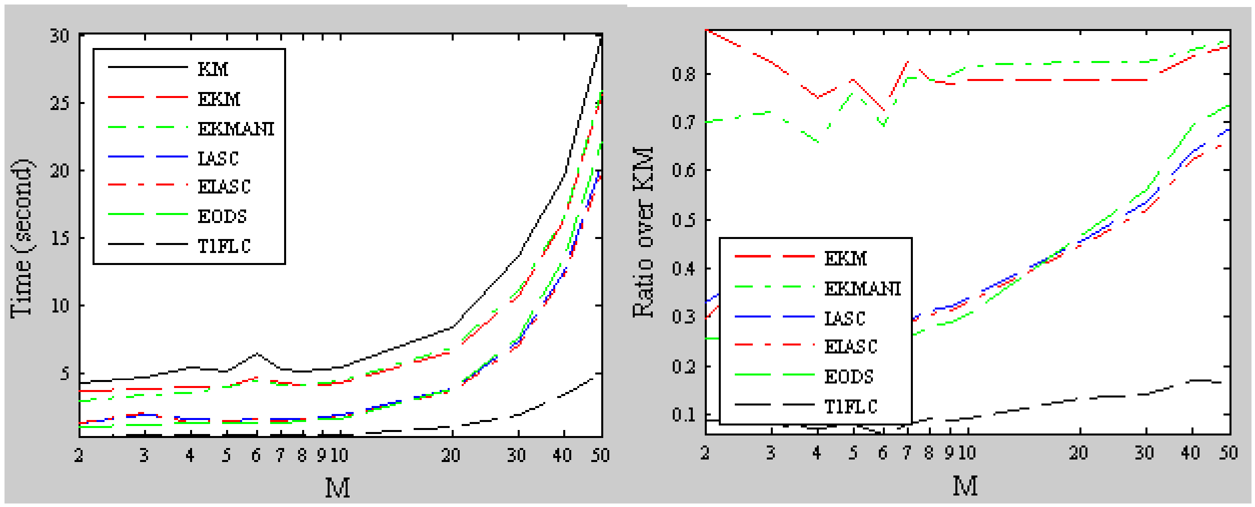
2. The EKM algorithms are faster than the EKMANI algorithm when M > 6.

3. Both the IASC and the EIASC algorithms are significantly faster than the KM, EKM, and EKMANI algorithms, especially when M is small (M ≤ 10, i.e., the rulebase has no more than 100 rules). The EIASC algorithms also outperform the IASC algorithms slightly.

4. The EODS algorithms are the fastest when M is small (M < 20, i.e., the rulebase has less than 400 rules), which is the case for most practical IT2 FLCs.

5. But when M ≥ 20, i.e., the rulebase has greater than 400 rules then EIASC algorithms are the fastest among all the enhanced KM algorithms.

6. Although the EODS algorithms are the fastest enhancement to the KM algorithms when M < 20, they are still about three times slower than the T1 FLC.



(a) (b)

Figure 3.5 Computational cost of the KM algorithms and their enhanced versions in control surface computation using Gaussian MFs. (a) Total computation time of the 100 control surfaces for different M (the number of IT2 FSs in each input domain). Note that $N = M^2$. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.2 Computational time using control surface for different enhanced KM algorithms when Gaussian IT2 FSs are used

M(N=M^2)	**KM**	**EKM**	**EKMANI**	**IASC**	**EIASC**	**EODS**	**T1**
M=2	4.1888	3.7303	2.9271	1.3830	1.2509	1.0737	0.35987
M=3	4.6254	3.8054	3.3238	1.9608	2.0128	1.2097	0.3662
M=4	5.3714	4.0249	3.5445	1.6380	1.3808	1.3428	0.3804
M=5	5.1446	4.0336	3.9280	1.4735	1.4020	1.3723	0.4166
M=6	6.4085	4.6385	4.4404	1.6197	1.6400	1.2685	0.3874
M=7	5.2310	4.3152	4.1215	1.5394	1.5078	1.3320	0.4198
M=8	5.1867	4.0719	4.08521	1.6299	1.5735	1.4533	0.4748
M=9	5.3369	4.1539	4.2391	1.7208	1.6652	1.5459	0.4620
M=10	5.4881	4.3192	4.4713	1.8621	1.8034	1.6740	0.5020
M=20	8.3490	6.5700	6.8539	3.7679	3.7029	3.8764	1.0882
M=30	13.6689	10.7487	11.2197	7.2939	7.1000	7.6408	1.9460
M=40	19.5182	16.3328	16.5318	12.4517	12.1406	13.5227	3.3036
M=50	30.0577	25.7453	26.0375	20.6037	19.9715	22.1326	4.9843

7. When M ≥ 8 EKM has the better performance than EKMANI algorithms.

8. When M=3 and M=6 the IASC has the better performance than EIASC.

9. Naturally the computational cost is reduced from EKM to EODS methods. We can consider that the EODS method is the fastest enhanced to the KM algorithms.

3.4.2 In control surface using Trapezoidal MFs

The previous experiment is repeated here using Trapezoidal MFs. There are many different methods to generate Trapezoidal IT2 FSs. In Fig 3.7 (a) the Trapezoidal MFs are generated for three IT2 FSs (M=3) in an input domain. Their corresponding control surface is represented in Fig 3.7 (b). The apexes of the UMFs were generated randomly under the constraint that for any point in the input domain, its firing levels on all UMFs add to 1. After the UMFs were generated, the LMF for each IT2 FS was also generated randomly with some constraints. Take the LMF of the middle IT2 FS as an example. e is a random number between a and b, f and g are two random numbers between b and c, i is a random number between c and d, and h is a random number in [0, 1]. This is shown in Fig 3.6.

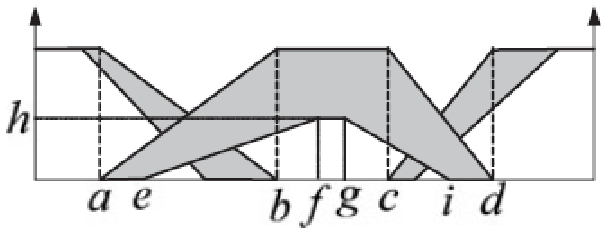


Figure 3.6 Trapezoidal IT2 FSs used in the experiments

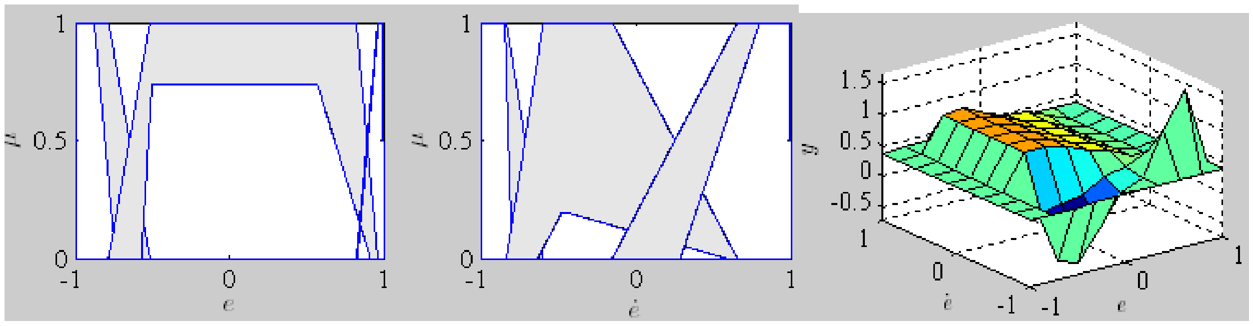
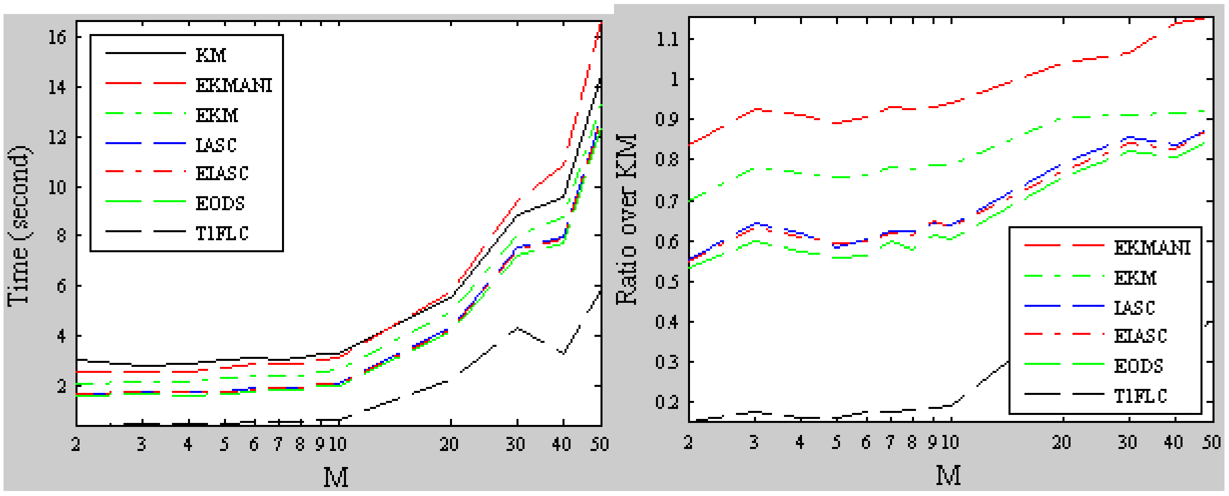


Figure 3.7 (a) A generated trapezoidal MFs (b) The corresponding control surface.

The experimental results for control surface using Trapezoidal MFs are shown in Fig 3.8 and Table 3.3. It is examined the following points from this experiment.

1. Except for the EKMANI algorithms, all other four enhancements are faster than the KM algorithms. However, the computational cost saving is not as large as that in Fig. 3.5 using Gaussian MFs. This is because at any time maximum two Trapezoidal MFs IT2 FSs are fired in



(a) (b)

Figure 3.8 Computational cost of the KM algorithms and their enhanced versions in control surface computation using trapezoidal MFs. (a) Total computation time of the 100 control surfaces for different *M* (the number of IT2 FSs in each input domain). Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.3 Computational time using control surface for different enhanced KM algorithms when Trapezoidal IT2 FSs are used

M	KM	EKM	EKMANI	IASC	EIASC	EODS	T1
M=2	3.0370	2.1229	2.5459	1.6821	1.6730	1.6240	0.4539
M=3	2.8069	2.1912	2.6044	1.8005	1.7746	1.6773	0.5011
M=4	2.8627	2.1990	2.6156	1.7640	1.7395	1.6458	0.4679
M=5	3.0447	2.3099	2.7206	1.7834	1.8140	1.6966	0.4931
M=6	3.1587	2.4010	2.8655	1.8989	1.8967	1.7782	0.5546
M=7	3.0827	2.4053	2.8756	1.9239	1.9131	1.8415	0.5507
M=8	3.1423	2.4359	2.9053	1.9616	1.9299	1.8202	0.5651
M=9	3.2944	2.6006	3.0746	2.1262	2.1282	2.0238	0.6224
M=10	3.3248	2.6236	3.1280	2.1285	2.1129	1.9990	0.6422
M=20	5.5313	5.0182	5.7608	4.3865	4.2679	4.1782	2.2772
M=30	8.8329	8.0677	9.4079	7.5783	7.4379	7.2479	4.3233
M=40	9.5498	8.7634	10.8780	7.9717	7.8861	7.6885	3.3161
M=50	14.4684	13.3412	16.6672	12.7730	12.7192	12.3588	5.8647

each input domain and hence at most four rules are fired. Therefore, even though M may be a very large number, the actual N used in TR is always no larger than four (on the other hand, for Gaussian MFs, the actual N used in TR is always equal to M^2). As a result, all algorithms converge very quickly.

2. The EKMANI algorithms are slower than the EKM algorithms. When M becomes large ($M \geq 20$), they are even slower than the KM algorithms. This suggests that when trapezoidal IT2 FSs are used, it may not be advantageous to initialize the switch points from previous TR results.

3. The IASC and EIASC algorithms again have almost the same speed.

4. The EODS algorithms are the fastest enhancement for all M presented here. Again, this is because at any time at most two trapezoidal IT2 FSs are fired in each input domain, and hence at most four rules are fired. Therefore, even though M may be a very large number, the actual N used in TR is always no larger than four. The EODS algorithms are very fast for small N, which is also noticeable from Fig. 3.5.

5. Although the EODS algorithms are the fastest enhancement to the KM algorithms, they are still about two to four times slower than the T1 FLC.

3.4.3 In Evolutionary Interval Type-2 Fuzzy Logic Controller using Gaussian MFs

It is also compared the computational cost of the KM algorithms and their five enhancements in IT2 FLC design using evolutionary computation, where the performance of a large number of (usually randomly generated) FLCs are evaluated. The following simple first-order plus dead-time plant is employed as the nominal system

$$G(s) = \frac{K}{\tau s + 1} e^{-Ls} = \frac{1}{10s + 1} e^{-2.5s} \qquad (3.1)$$

The goal is to design an IT2 fuzzy PI controller

$$\dot{u} = k_P \dot{e} + k_I e \qquad (3.2)$$

where $\dot{u}$ is the change in control signal, e is the error, $\dot{e}$ is the change of error, and k_P and k_I are proportional and integral gains

First, it is considered that there are M Gaussian IT2 FSs in each domain (e and $\dot{e}$), and each IT2 FSs is evaluated by three parameters (one mean, m_m, and two standard deviations, σ_m^1 and σ_m^2, m = 1, 2, . . .,M). Each of the M^2 rule consequents is represented by a crisp number y^n, n = 1, . . . , M^2 . Then, for each input pair (e, $\dot{e}$), all N = M^2 rules are fired, and a TR algorithm is needed to compute the output of the IT2 FLC. The population consisted of 100 randomly generated IT2 FLCs (all m_m, σ_m^1 , σ_m^2 , and y^n were generated randomly), and the performance of each FLC was evaluated by a step response in the first 100 s with sampling frequency 1 Hz. We recorded the time that the five algorithms were used to perform these 100 × 100 = 10 000 TRs. M = {2, 3, . . . , 10, 20, . . . , 50} were used. The results are shown in Fig. 3.9 and the Table 3.4. It is observed the following points.

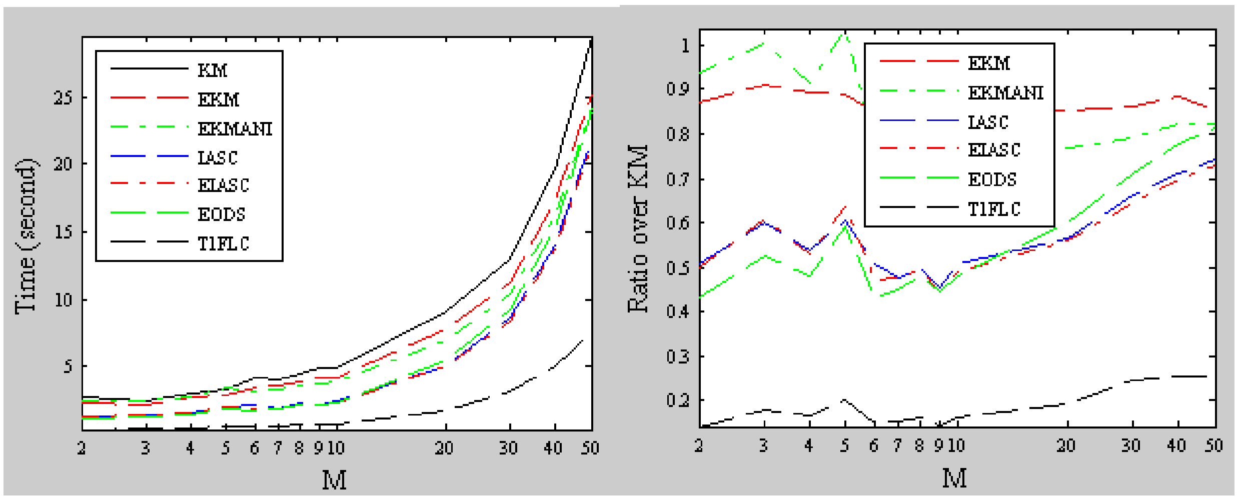
1. Generally, all five enhanced algorithms are faster than the KM algorithms.

2. The EKMANI algorithms are faster than the EKM algorithms when M $\geq$ 4.

3. Both the IASC and the EIASC algorithms are significantly faster than the KM, EKM, and EKMANI algorithms, especially when M is small (M ≤ 20, i.e., the rulebase has no more than 400 rules). The EIASC algorithms also outperform the IASC algorithms slightly.

4. The EODS algorithms are the fastest when M is small (M ≤ 10, i.e., the rulebase has no more than 100 rules), which is the case for most practical IT2 FLCs.

5. Although the EODS algorithms are the fastest enhancement to the KM algorithms when M ≤ 10, they are still about three times slower than the T1 FLC.



(a) (b)

Figure 3.9 Computational cost of the KM algorithms and their enhanced versions in evolutionary IT2 FLC design using Gaussian MFs. (a) Total computation time of the 100 IT2 FLCs for different M (the number of IT2 FSs in each input domain). Note that N = M^2. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.4 Computational time using FLC for different enhanced KM algorithms when Gaussian IT2 FSs are used

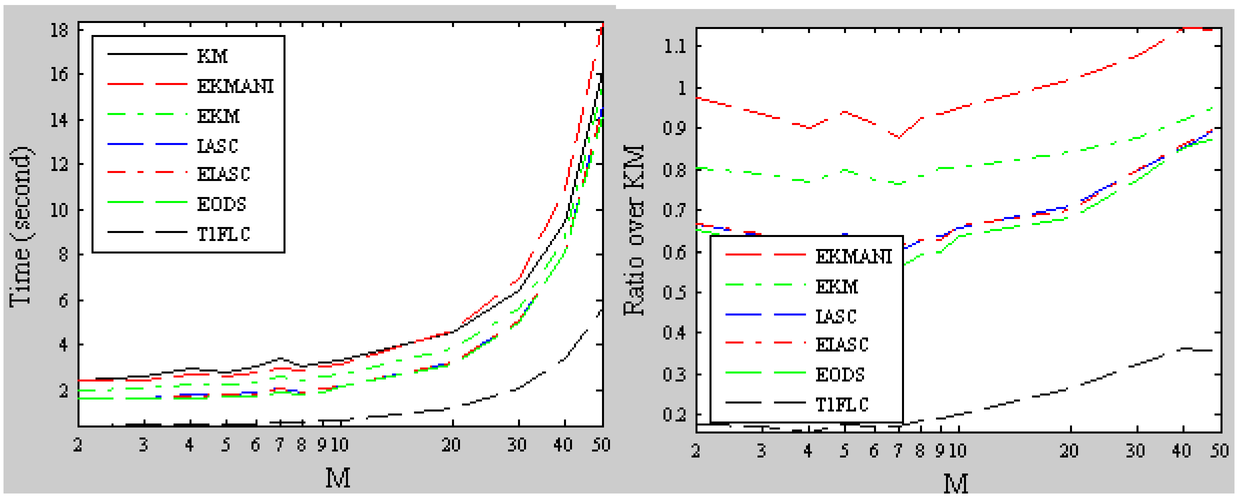
M(N=M^2)	KM	EKM	EKMANI	IASC	EIASC	EODS	T1
M=2	2.7037	2.3567	2.5380	1.3757	1.3526	1.1662	0.3844
M=3	2.4619	2.2473	2.4753	1.4781	1.5000	1.2911	0.4454
M=4	3.0086	2.6920	2.7566	1.6235	1.5925	1.4452	0.5034
M=5	3.2935	2.9275	3.4164	1.9890	2.0997	1.9453	0.6635
M=6	4.1713	3.5485	3.2342	2.1260	1.9673	1.8085	0.6157
M=7	4.1033	3.6267	3.2925	1.9572	1.9585	1.8481	0.6229
M=8	4.5309	3.8909	3.6093	2.2623	2.2450	2.1875	0.7444
M=9	4.9256	4.2083	3.6998	2.2336	2.2005	2.1956	0.7191
M=10	4.8728	4.1809	3.8768	2.4748	2.3894	2.3488	0.7903
M=20	9.0852	7.7394	7.0031	5.1340	5.1045	5.4582	1.7709
M=30	13.0383	11.2458	10.3315	8.6529	8.4259	9.2786	3.1948
M=40	19.6307	17.3318	16.1746	13.9594	13.6653	15.3124	5.0243
M=50	29.4789	25.1959	24.1811	21.9164	21.5055	23.9459	7.5410

6. When M ≥ 10 then the computational cost of EIASC is only reduced and it becomes the fastest among all other algorithms.

7. IASC has better performance than EIASC when M = 3, 5, 7 only.

3.4.4 In Evolutionary Interval Type-2 Fuzzy Logic Controller using Trapezoidal MFs

It may be also compared the all enhanced to KM algorithms in Evolutionary Interval Type-2 Fuzzy Logic controller using Trapezoidal MFs. All the comparisons using above methods are presented in Fig 3.10 and Table 3.5.



(a) (b)

Figure 3.10 Computational cost of the KM algorithms and their enhanced versions in evolutionary IT2 FLC design using trapezoidal MFs. (a) Total computation time of the 100 IT2 FLCs for different *M* (the number of IT2 FSs in each input domain). Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.5 Computational time using FLC for different enhanced KM algorithms when Trapezoidal IT2 FSs are used

M	KM	EKM	EKMANI	IASC	EIASC	EODS	T1
M=2	2.4726	1.9807	2.4035	1.6492	1.6449	1.6167	0.4333
M=3	2.6062	2.0507	2.4365	1.6636	1.6750	1.6039	0.4522
M=4	2.9515	2.2723	2.6525	1.7851	1.7568	1.6637	0.4729
M=5	2.8068	2.2401	2.6368	1.8024	1.7772	1.6972	0.4976
M=6	3.0184	2.3305	2.7436	1.8668	1.8477	1.7456	0.5162
M=7	3.3895	2.5858	2.9717	2.0312	2.0825	1.8920	0.5865
M=8	3.0935	2.4292	2.8659	1.9370	1.9402	1.8412	0.5750
M=9	3.2448	2.6006	3.0344	2.0649	2.0402	1.9364	0.6309
M=10	3.3330	2.6701	3.1695	2.1931	2.2035	2.1308	0.6800
M=20	4.5927	3.8775	4.6749	3.2611	3.2221	3.1316	1.2219
M=30	6.4230	5.6419	6.9231	5.1137	5.1338	4.9711	2.0933
M=40	9.4718	8.7085	10.8723	8.0705	8.1427	8.0708	3.4379
M=50	16.0956	15.4792	18.3558	14.5773	14.5701	14.1683	5.6587

1. The entire enhanced versions are faster than KM algorithm.

2. But they are still slower than T1 FLC.

3. It is also observed that the computational cost using EKMANI method is higher than EKM method or any value of M.

4. EIASC algorithm has the almost same computational speed as IASC algorithm.

5. EODS method is the fastest method among all the enhanced version of KM algorithms for any value of M. But this method is still 3 to 4 times slower than T1 FLC.

From the above experiments it is examined that among the KM algorithms and their five enhanced versions, the EODS algorithms seem to be the fastest to use when there are less than 100 rules fired each time, which is usually true in practice. However, they are much more complex than the EIASC algorithms, which are only slightly ($\leq$ 1.2 times) slower than the EODS algorithms for practical IT2 FLCs. For the point of ease in understanding and implementation, the EIASC algorithms may be preferred in practice. But all the enhanced to KM algorithms are three times slower than T1 FLC.

We can also check whether removing the zero weight elements in EIASC algorithms can improve the speed using both Gaussian MFs and Trapezoidal MFs. First we consider a normal EIASC algorithms and a removed zero weight elements in EIASC2 algorithm. The comparison of EIASC and EIASC2 algorithms using Gaussian MFs and Trapezoidal MFs are shown in Fig 3.11 and Fig 3.12 respectively.

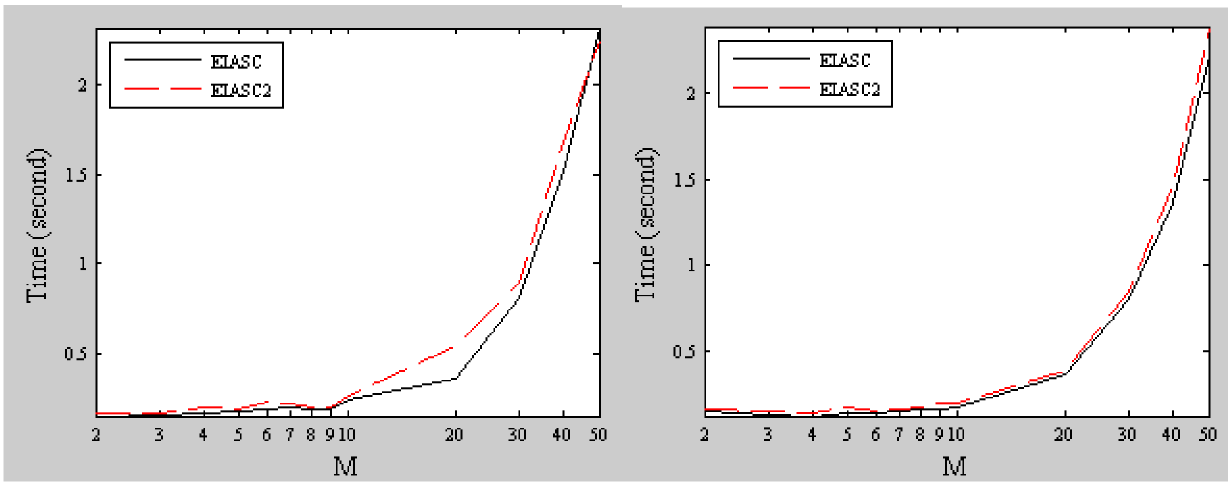


Figure 3.11 Comparison of EIASC and EIASC2 algorithms using Gaussian MFs

All the experimented results by using both Gaussian MFs and Trapezoidal MFs are illustrated in Table 3.6.

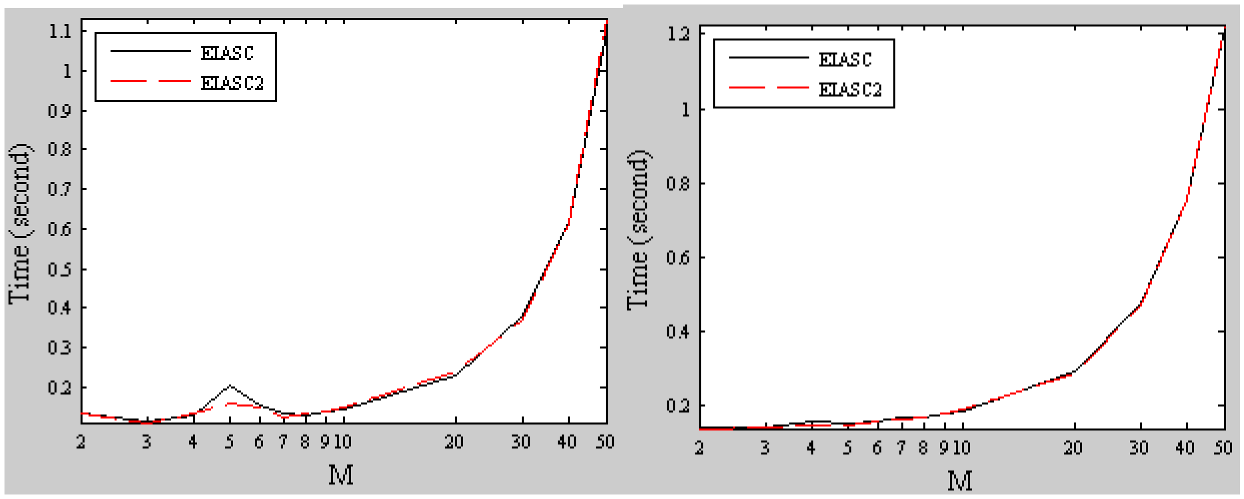


Figure 3.12 Comparison of EIASC and EIASC2 algorithms using Trapezoidal MFs

Table 3.6 Computational time for EIASC and EIASC2 algorithms using Gaussian MFs and Trapezoidal MFs

M	EIASC Time using Gaussian MFs	EIASC2 Time using Gaussian MFs	EIASC Time using Trapezoidal MFs	EIASC2 Time using Trapezoidal MFs
M=2	0.148748234761681	0.166850535473084	0.141499548126927	0.139612436776182
M=3	0.127866809887849	0.155788235655649	0.143431078530931	0.142345186329547
M=4	0.126325552549277	0.145432729578759	0.157542089846615	0.149026203050946
M=5	0.140725706758820	0.172621380650334	0.151714254185937	0.149543028513400
M=6	0.145103916838593	0.158206699454819	0.161020744256602	0.159656325035724
M=7	0.148762482382537	0.165906560750039	0.168595449980375	0.166211068725215
M=8	0.162969595297726	0.174015412573386	0.170095081916836	0.168349888044430
M=9	0.162812033372957	0.194046729402759	0.179560251372730	0.178706790946894
M=10	0.175986892188812	0.197442133008525	0.187527465082853	0.190584836899662
M=20	0.372238371077888	0.393829383343414	0.294087377026969	0.286668557037277
M=30	0.800464304820864	0.847323891723669	0.475043057148325	0.469598510425208
M=40	1.35801617244650	1.46220539202608	0.757581480327807	0.756467651614940
M=50	2.22525600320711	2.38455418216561	1.21622412904433	1.22586837153884

From the above table we get that the computational time of EIASC is less than EIASC2 algorithms by using both Gaussian MFs. But the computational cost of EIASC method is almost similar to the computational cost of EIASC2 method using Trapezoidal MFs. So it should be concluded that removing the zero weight elements in EIASC algorithms cannot reduce the computational cost. Hence EIASC algorithms are faster than EIASC2 algorithms. So we can consider only EIASC algorithm. It can be observed that computational time of EIASC using Trapezoidal MFs is less than the computational time using Gaussian MFs. It should be also noticed that computational cost for removing zero weight elements in EIASC method using Trapezoidal MFs is much less than the computational cost using Gaussian MFs for any value of M.

3.5 Computational cost comparison of eleven alternative type reduction algorithms

3.5.1 In control surface using Gaussian MFs

A comparison of the computational cost of the eleven alternative type reduction (TR) algorithms and KM algorithms in control surface computation using Gaussian MFs is shown in Fig. 3.13. The computational time (seconds) of all these methods are illustrated in Table 3.7. The DY method is not included in this section because its computational cost is much higher than others and increases exponentially with respect to N. We can observe from Fig. 3.13 and Table 3.7 the following points.

1. All ten alternative TR algorithms in Fig. 3.13 are faster than the KM algorithms, especially when M is small, e.g., $M < 10$.

2. The WT, NT, Liang–Mendel (LM), and BMM methods have similar computational cost, and generally, they are faster than other alternative TR algorithms.

3. WT and NT methods are the fastest among all other alternative TR methods.

4. When $M < 7$ Collapsing method is fast than BMM method.

5. When $M < 10$ the LYZ algorithm has less computational cost than LM algorithm.

6. The Geometric and Gorzalczany Methods have the almost similar computational cost.

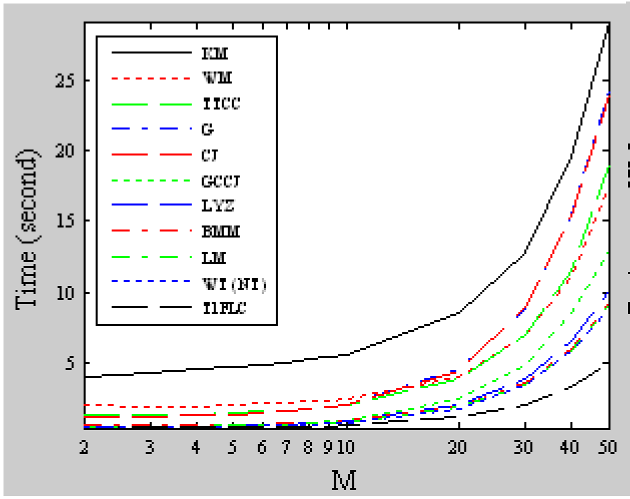
7. The WT and NT methods are about 1.2–1.7 times slower than a T1 FLC.

8. TTCC method has the high speed of operation than Geometric method when $M \geq 10$.

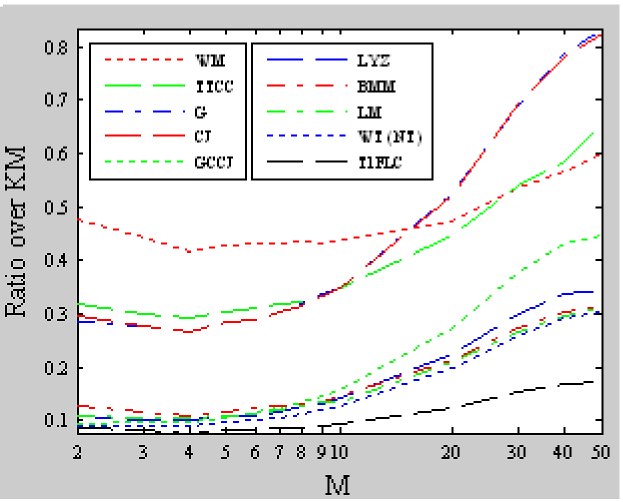
9. WT and NT method is also faster than the fastest enhanced to the KM algorithms, EODS method.

We can observe from the Table 3.7 that the WT, NT, LM, and BMM methods are much faster than the EODS algorithms, the fastest enhancement to the KM algorithms. However, there is an important difference between the methods presented in this section and those in the previous section: All the enhanced versions of the KM algorithms introduced in Section 2.5 give exactly the same outputs as the original KM algorithms, which have some fundamentally different characteristics from T1 FLCs. On the other hand, the alternative TR algorithms presented in this section have very different characteristics from the original KM algorithms, and hence their outputs are different.

All the computational cost comparison in alternative TR methods is done using Gaussian MFs. It can be seen that if the trapezoidal MFs are used the computational cost saving of all algorithms, including the T1 FLC, over the KM algorithms is not as large as that in the Gaussian MF case. The WT and NT method is still the fastest; however, the GCCJ method is equally fast. This is because when a very small number of rules are fired, the time used to construct the closed polygon in the GCCJ method is negligible.



(a)



(b)

Figure. 3.13 Computational cost of the alternative TR algorithms in control surface computation using Gaussian MFs. (a) Total computation time of the 100 IT2 FLCs for different M (the number of IT2 FSs in each input domain). Note that N = M^2. (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

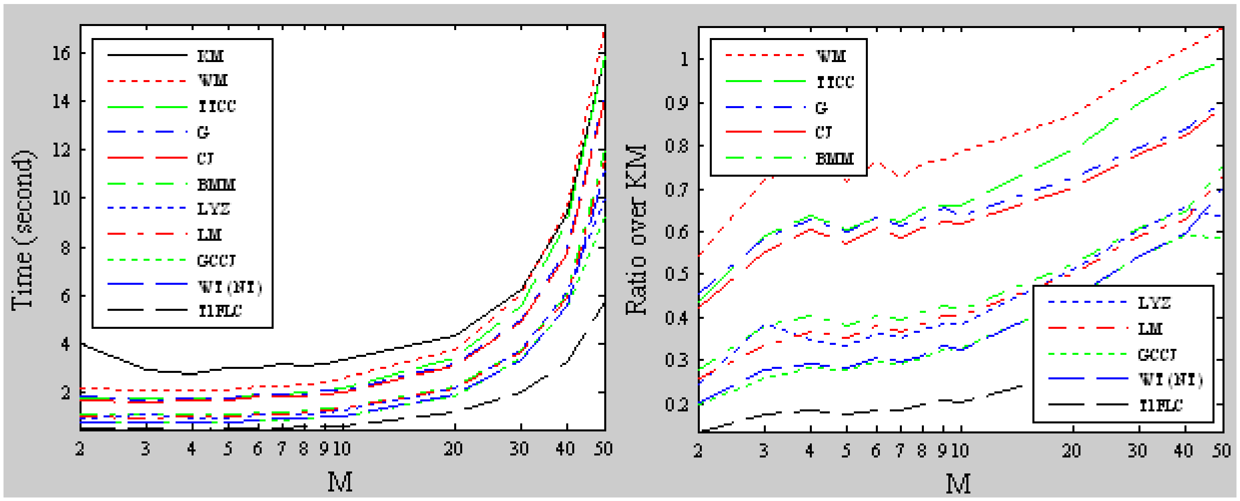
Table 3.7 Computational time using control surface for alternative type-reduction methods when Gaussian IT2 FSs are used

KM	**UB**	**WT& NT**	**BMM**	**TTCC**	**GCCJC**	**CJG**	**T1**	**G**	**LM**	**LYZ**	**EODS**
4.0196	1.9122	0.3685	0.5124	1.2792	0.3834	1.1887	0.3425	1.1411	0.4444	0.4440	1.0876
4.2340	1.8716	0.3871	0.4861	1.2754	0.4069	1.17583	0.3499	1.1783	0.4440	0.4373	1.0526
4.5622	1.9017	0.4087	0.5040	1.3282	0.4499	1.2244	0.3524	1.2111	0.4810	0.4627	1.1122
4.6244	1.9877	0.4471	0.5352	1.3995	0.4922	1.3174	0.3751	1.3138	0.5094	0.4960	1.1582
4.8607	2.1060	0.4949	0.6055	1.5121	0.5590	1.3975	0.4048	1.4130	0.5524	0.5315	1.2556
4.9701	2.1401	0.5218	0.6347	1.5929	0.6256	1.5065	0.4252	1.5041	0.5981	0.5836	1.3340
5.1902	2.2607	0.5904	0.6916	1.6828	0.6990	1.6296	0.4579	1.6569	0.6548	0.6541	1.4360
5.3277	2.3059	0.6396	0.7293	1.7995	0.7745	1.7805	0.4806	1.8099	0.7144	0.7045	1.5649
5.4798	2.4154	0.7029	0.8055	1.9111	0.8710	1.9121	0.5215	1.9189	0.7514	0.7874	1.6887
8.5598	4.0346	1.6913	1.8459	3.8265	2.3443	4.4618	1.0666	4.5027	1.8143	1.9372	3.8041
12.8058	6.8867	3.33865	3.4881	6.9061	4.8417	8.8867	1.9653	8.8662	3.4244	3.8285	7.6519
19.5961	11.1017	5.7463	5.9401	11.4623	8.4900	15.2990	3.2936	15.4291	5.7977	6.5926	13.1116
29.0459	17.4962	8.8431	9.2022	19.0415	12.9372	23.9458	4.9875	24.2543	9.0364	10.0786	22.4255

3.5.2 In control surface using Trapezoidal MFs

The previous experiments are repeated for Trapezoidal IT2 FSs. The results are shown in Table 3.8 and a pictorial overview is presented in Fig 3.14. From that table and figure we can conclude the following points.

1. The computational cost saving of all algorithms, including the T1 FLC, over the KM algorithms is not as large as that in the Gaussian MF case. This is because for trapezoidal MFs at most four rules are fired at any time, so all algorithms converge very quickly.



(a) (b)

Figure 3.14 Computational cost of the alternative TR algorithms in control surface computation using trapezoidal MFs. (a) Total computation time of the 100 IT2 FLCs for different M (the number of IT2 FSs in each input domain). Note that the rulebase has N = M^2 rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.8 Computational time using control surface for alternative type-reduction methods when Trapezoidal IT2 FSs are used.

KM	UB	WT& NT	BMM	TTCC	GCCJC	CJG	T1	G	LM	LYZ	EODS
4.0371	2.1867	0.8171	1.1221	1.7594	0.8065	1.7055	0.5435	1.8383	1.04356	0.9922	1.6798
2.9776	2.1433	0.8196	1.1261	1.7588	0.7761	1.6500	0.5183	1.7477	0.99173	1.1525	1.5777
2.8094	2.1277	0.8119	1.1294	1.7922	0.7927	1.6988	0.5154	1.7650	1.03175	0.9733	1.6161
2.9986	2.1544	0.8429	1.1454	1.8173	0.8330	1.7169	0.5280	1.8001	1.0546	1.0024	1.6476
3.0396	2.3145	0.9253	1.2335	1.9174	0.9036	1.8530	0.5650	1.9300	1.1538	1.1019	1.7791
3.1651	2.3041	0.9317	1.2543	1.9687	0.9230	1.8488	0.5847	1.9415	1.1637	1.1201	1.7858
3.1089	2.3587	0.9665	1.2846	2.0406	0.9561	1.8990	0.6077	1.9494	1.1908	1.1495	1.9409
3.2127	2.4673	1.0673	1.3656	2.1165	1.0417	1.9984	0.6627	2.1093	1.3031	1.2359	1.9205
3.3279	2.6221	1.0807	1.3892	2.2060	1.0877	2.0648	0.6715	2.1281	1.3395	1.2740	1.9781
4.3556	3.7965	1.9328	2.2818	3.4414	1.9115	3.0714	1.1841	3.1630	2.1829	2.2267	3.0135
6.2515	6.0766	3.3905	3.7980	5.6142	3.4146	4.8946	2.0358	4.9797	3.6916	3.7864	4.8928
9.3810	9.6322	5.5915	6.0607	9.0736	5.5387	7.7234	3.3077	7.8821	5.8725	6.1501	7.9823
15.8950	17.1588	11.1863	12.0008	15.8420	9.2842	14.1329	5.7110	14.333	11.6124	10.0591	13.9730

2. The WT and NT methods are still the fastest. However, the GCCJC method is equally fast. This is because when a very small number of rules are fired, the time used to construct the closed polygon in the GCCJC method is negligible.

3. The computational cost of UB method is higher than KM method when $M > 30$.

4. The computational cost of LM and LYZ methods are almost identical.

From the above experiments we get that the WT, NT, GCCJC and BMM methods are much faster than the EODS algorithms, which are the fastest enhancement to the KM algorithms.

3.5.3 In Evolutionary Interval Type-2 Fuzzy Logic Controller using Gaussian MFs

It is also compared the computational cost of the alternative TR algorithms in evolutionary FLC design. The results are shown in Fig 3.15, and the Table 3.9, for Gaussian IT2 FSs. All alternative TR algorithms are much faster than the KM algorithms when Gaussian IT2 FSs are used. However, it may be also shown that their computational cost savings are not so large when trapezoidal IT2 FSs are used. The DY method also is not included in this comparison because its computational cost is much higher than others and increases exponentially with respect to N. From Fig 3.15 and Table 3.9 it is examined that

1. WT-NT, LM, and BMM methods are much faster than the EODS algorithms, the fastest enhancement to the KM algorithms.
2. WT and NT algorithm has the same computational cost as LM algorithm.
3. When $M \leq 20$ EODS method is faster than UB method.
4. LM method is always fast than Geometric, Gorzalczany, BMM and LYZ method.
5. In this experiment WT and NT methods are the fastest among all other alternative TR methods.
6. But the WT and NT methods are about 1.2–1.7 times slower than a T1 FLC.
7. The LM method is also faster than Collapsing method when $M > 5$.

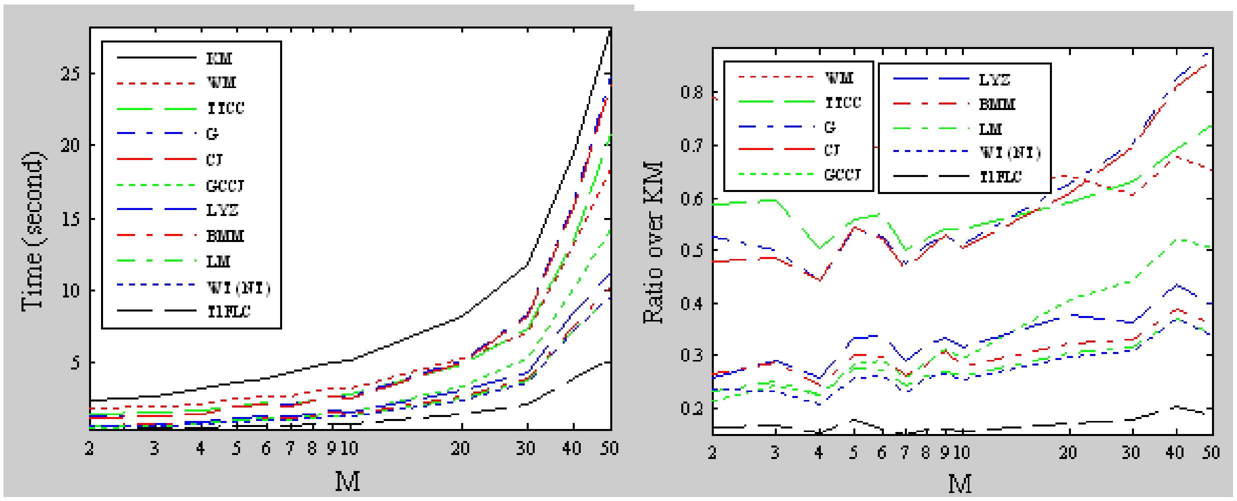
3.5.4 In Evolutionary Interval Type-2 Fuzzy Logic Controller using Trapezoidal MFs

In this section the computational cost comparison is examined in evolutionary interval type-2 fuzzy logic controller using Trapezoidal MFs. From figure 3.16 and Table 3.10, we get the following points.

1. WT-NT, LM, and BMM methods are much faster than the EODS algorithms, the fastest enhancement to the KM algorithms.
2. Here LYZ method has the better performance than LM method when $M \leq 10$.
3. LM method is always fast than Geometric, Gorzalczany and BMM method.
4. In this experiment also WT and NT methods are the fastest among all other alternative TR methods.

5. But the WT and NT methods are about 1.2–1.7 times slower than a T1 FLC.
6. The computational cost of Collapsing method is almost identical to the WT-NT method when M >3.

From the above experiments using both Gaussian and Trapezoidal MFs it is examined that WT and NT method is two times faster than the fastest enhanced to KM algorithms, EODS method.



(a) (b)

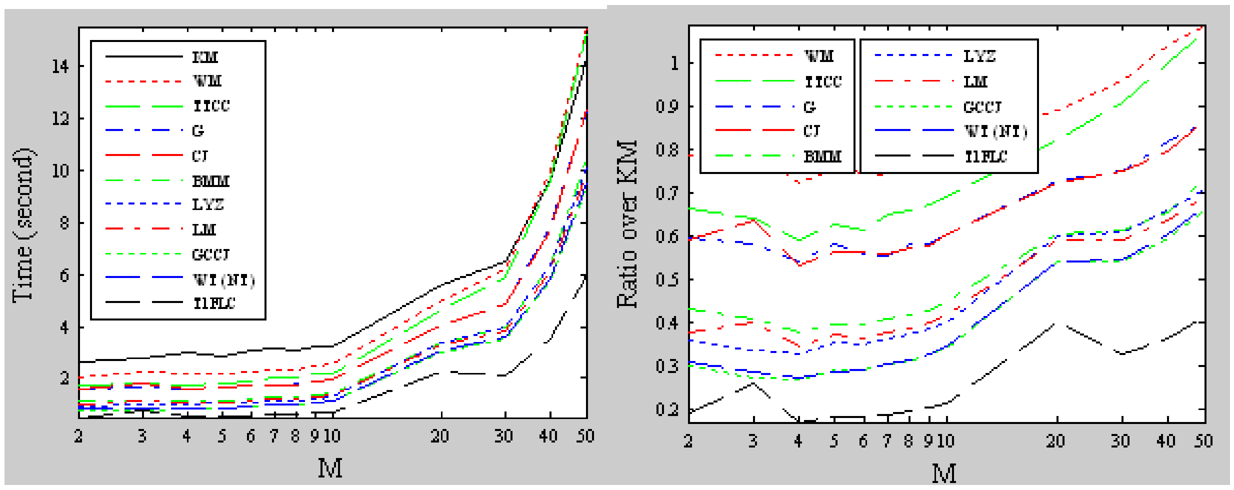
Fig.3.15 Computational cost of the alternative TR algorithms in evolutionary IT2 FLC design using Gaussian MFs. (a) Total computation time of the 100 IT2 FLCs for different M: the number of IT2 FSs in each input domain. Note that N = M^2 . (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.9 Computational time using FLC for alternative type-reduction methods when Gaussian IT2 FSs are used

KM	UB	WT& NT	BMM	TTCC	GCCJC	CJG	T1	G	LM	LYZ	EODS
2.4190	1.9095	0.5688	0.6431	1.4224	0.5241	1.1616	0.3946	1.2704	0.5661	0.6254	1.1957
2.6602	2.0185	0.6175	0.7679	1.5778	0.6466	1.2924	0.4427	1.3326	0.6691	0.7739	1.3468
3.2709	2.0951	0.6812	0.7977	1.6535	0.7398	1.4493	0.4967	1.4493	0.7356	0.8459	1.4063
3.5831	2.5746	0.9279	1.0831	2.0003	1.0209	1.9526	0.6389	1.9413	0.9868	1.1953	1.7684
3.9564	2.7187	1.0296	1.1849	2.2514	1.1535	2.0707	0.6412	2.0831	1.0729	1.3332	2.0741
4.3650	2.6948	1.0062	1.1421	2.1809	1.1243	2.0267	0.6623	2.0763	1.0588	1.2649	1.9165
4.6680	3.0851	1.2055	1.3250	2.4390	1.3457	2.3312	0.7466	2.3811	1.2166	1.4934	2.1575
4.9770	3.2216	1.3154	1.5330	2.6976	1.5570	2.6414	0.8074	2.6242	1.3401	1.66538	2.4705
5.1059	3.1878	1.2968	1.4340	2.7525	1.5014	2.5689	0.8047	2.6060	1.3310	1.6146	2.3934
8.1549	5.2264	2.4414	2.6370	4.8085	3.3076	4.9546	1.4008	5.1226	2.5023	3.0662	4.7125
11.7607	7.1229	3.6147	3.8844	7.4050	5.2209	8.1959	2.1099	8.2619	3.7346	4.2770	7.9734
19.4454	13.1920	7.1942	7.5540	13.4296	10.1395	15.7530	3.9558	16.0734	7.1852	8.4882	15.1188
28.2340	18.4206	9.4992	10.1844	20.9045	14.2464	24.3033	5.2734	24.9076	9.6909	11.1536	23.8870

3.6 Comparison of defuzzified outputs of alternative TR algorithms

The alternative TR algorithms, generally, present different defuzzified outputs from the KM algorithms and enhanced to the KM algorithms. Since KM algorithm is the most popular method to calculate the defuzzified outputs in type reduction algorithms. Many theoretical and practical results are based on the KM algorithm. So it is very interesting to see how the defuzzified results of all TR algorithms are closed to the KM methods. The results are obtained in the previous experiments to perform some qualitative comparison, and this should also give us some useful insight. The statistics of the absolute difference



(a) (b)

Figure 3.16 Computational cost of the alternative TR algorithms in evolutionary IT2 FLC design using trapezoidal MFs.(a) Total computation time of the 100 IT2 FLCs for different M: the number of IT2 FSs in each input domain. Note that N = M^2. (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.10 Computational time using FLC for alternative type-reduction methods when Trapezoidal IT2 FSs are used

KM	UB	WT& NT	BMM	TTCC	GCCJC	CJG	T1	G	LM	LYZ	EODS
2.6163	2.0554	0.8138	1.1336	1.7333	0.7898	1.5531	0.5003	1.5558	0.9891	0.9427	1.7003
2.8076	2.2232	0.8146	1.1472	1.7995	0.7705	1.7905	0.7380	1.6371	1.1254	0.9490	1.5591
2.9641	2.1456	0.8083	1.1235	1.7599	0.8051	1.5757	0.5112	1.6032	1.0282	0.9777	1.6012
2.8722	2.1695	0.8281	1.1374	1.8101	0.8393	1.6249	0.5321	1.6731	1.0695	1.0235	1.6794
3.0658	2.2863	0.8915	1.2140	1.8835	0.8978	1.7253	0.5700	1.7175	1.1190	1.0756	1.7325
3.1219	2.3160	0.9592	1.2752	2.0296	0.9548	1.7422	0.5919	1.7380	1.1792	1.1330	1.7544
3.0592	2.3465	0.9635	1.2813	2.0173	0.9559	1.7573	0.6083	1.7773	1.1889	1.1451	1.7955
3.1934	2.4497	1.0545	1.3715	2.1542	1.0480	1.8518	0.6539	1.8666	1.2759	1.2335	1.8871
3.1940	2.5772	1.1098	1.4377	2.2104	1.0980	1.9337	0.6873	1.9375	1.3366	1.2856	1.9949
5.5929	4.9954	3.0366	3.3976	4.6064	3.0278	4.0579	2.2449	4.0649	3.3112	3.3575	4.2039
6.4867	6.2194	3.5547	3.9863	5.9041	3.5258	4.8565	2.1356	4.8684	3.8380	3.9501	4.9781
9.6303	10.0264	5.8339	6.3126	9.6208	5.7468	7.6552	3.5177	7.8734	6.1317	6.3469	7.9124
14.2847	15.5459	9.5287	10.4920	15.3042	9.4127	12.3798	5.9457	12.3358	9.9054	10.1253	12.3418

between the output of the KM TR method and alternative TR methods are shown in Table 3.11. The graphical representation of this difference is presented in APPENDIX B. It should be observed the following points.

1. The WM method gives the closest approximation to the KM TR method.

2. Generally, the G, CJ, TTCC, BMM, NT, WT, and GCCJ methods give better approximation to the KM TR method than a T1 FLS, in which the UMFs of the corresponding IT2 FSs are used as its MFs.

3. The output of the LM method is significantly different from other methods because it is an unnormalized method.

4. There is a huge difference between the output of the LYZ method and all other methods.

Table 3.11 Statistics of the absolute difference between the outputs of alternative TR algorithms and those of the KM algorithms.

TR Algorithm	**Control Surface Computation using Gaussian MFs**		**Evolutionary IT2 FLC Design using Gaussian MFs**	
	Mean	Std	Mean	Std
WM	0.043	0.087	0.035	0.075
G	0.169	0.185	0.103	0.155
CJ	0.235	0.251	0.132	0.192
TTCC	0.062	0.148	0.148	0.238
BMM	0.065	0.153	0.153	0.243
WT-NT	0.065	0.157	0.158	0.257
GCCJ	0.066	0.163	0.166	0.266
T1	0.097	0.162	0.174	0.253
LM	2.702	4.536	1.164	2.815
LYZ	∞	∞	∞	∞

The alternative TR methods give different output from the KM TR method, but it does not mean that they have poor performance. In fact, many of them have demonstrated good performance in the literature. It only means that the MFs need to be retuned when a different TR method is used.

From the above tables it is examined that the WT, NT, LM, BMM, LYZ, and GCCJ methods consistently outperform the EODS algorithms, the fastest KM algorithm based type-reducer. Among them, the WT and NT methods seem to be the fastest alternative TR algorithms. The LM and LYZ methods give significantly different results from the KM algorithms. Among the eleven alternative TR methods, the BMM method has the highest robustness and stability which are examined.

3.7 Computational cost comparison of the simplified IT2 FSs

3.7.1 In control surface using Gaussian MFs

The computational cost of the simplified IT2 FLCs with "full" IT2 FLCs (whose all MFs are IT2 FSs) in control surface computation is compared in this section. Four TR approaches are considered.

1) The standard KM type-reducer

2) The EODS algorithms, which are the fastest enhancement to the KM algorithms

3) The WT (NT) method, which is the fastest alternative TR algorithm

4) The EIASC algorithms, which are the simplest enhancement to the KM algorithms.

The computational cost of the corresponding T1 FLCs is also recorded.

When Gaussian IT2 FSs are used and the simplified IT2 FLC has only the center MF in each input domain as IT2 FS, the results are shown in Fig.3.17 and the Table 3.12. The experimental setup was the same as that in Section 3.4.1. Observe from Fig. 3.17 the following points are concluded.

1. When M is small, e.g., when there are fewer than ten MFs in each input domain, the computational cost of the simplified IT2 FLC is higher than the corresponding full IT2 FLC (KMs versus KMf, EODSs versus EODSf, and WTs versus WTf) because of the extra effort in constructing Y_*^n in (2.48) and F^{N+1} in (2.49). When M gets larger, the simplified IT2 FLC becomes faster than the full IT2 FLC because the computational cost saving offered by the simplified structure outweighs the extra effort to construct Y_*^n and F^{N+1}.

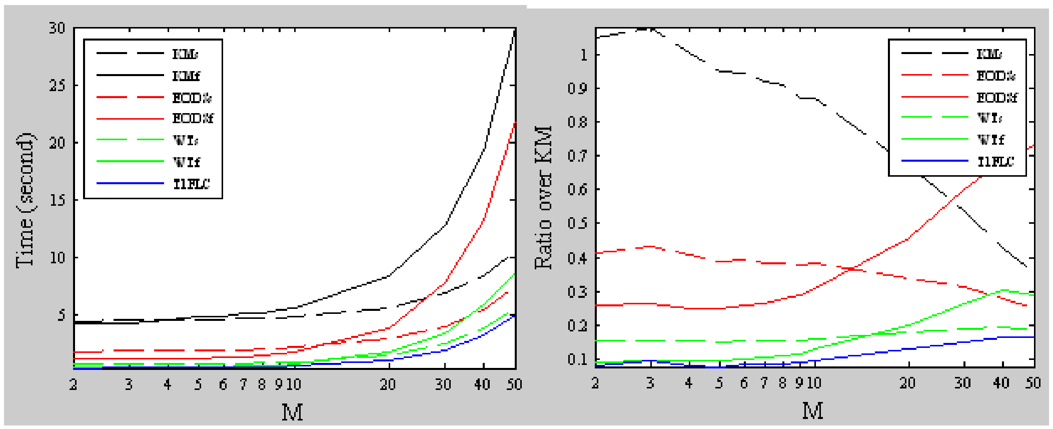
2. When M becomes larger, the speed of the simplified IT2 FLC, regardless of the TR method, approaches the T1 FLC because most part of the simplified IT2 FLC is essentially a T1 FLC.

3. But these simplified IT2 FLCs are still slower than T1 FLC for any value of M.

4. Among the all simplified and full IT2 FLCs methods simplified WT and NT method is the fastest method.

5. Simplified WT-NT method has the less computational cost than its full version when the fired rule is so large, i.e., $M \geq 20$.

6. Simplified WT-NT method has high speed than Simplified EODS method for any value of M.



(a) (b)

Fig. 3.17 Computational cost of the simplified IT2 FLC in control surface computation using Gaussian IT2 FSs. (a) Total computation time of the 100 IT2 FLCs for different M: the number of IT2 FSs in each input domain. Note that N = M^2. (b) Ratio of the computation time of different algorithms to the KM algorithms. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

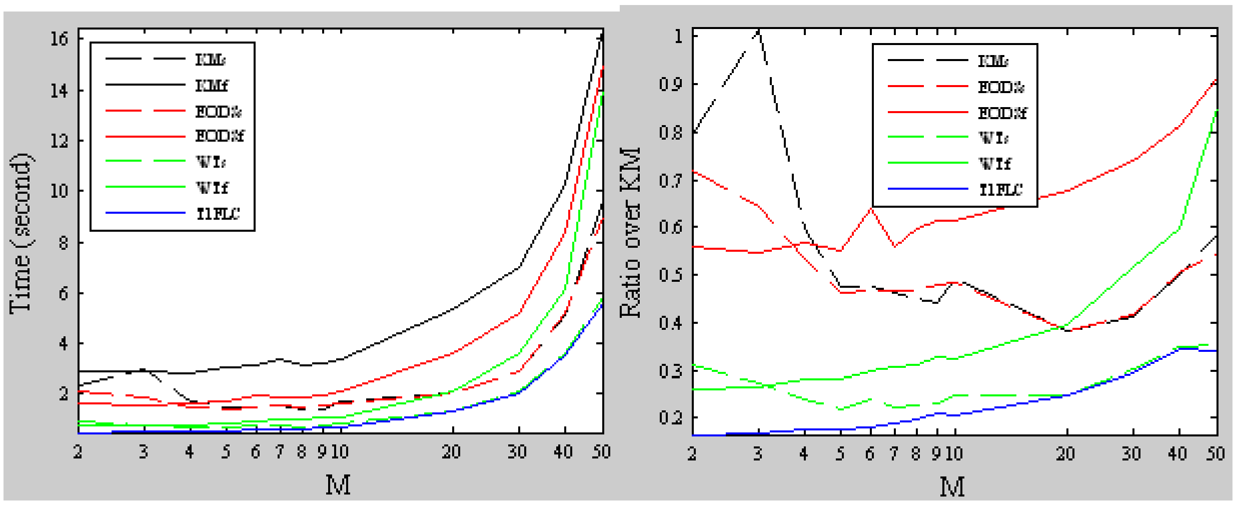
Table 3.12 Computational time using control surface for simplified IT2 FSs when Gaussian IT2 FSs are used

M ($N=M^2$)	**Full KM**	**Full EIASC**	**Full EODS**	**Full WT&NT**	**Simple KM**	**Simple EIASC**	**Simple EODS**	**Simple WT&NT**	**T1**
M=2	4.1742	1.2469	1.0807	0.3733	4.3820	1.9287	1.7264	0.6399	0.3317
M=3	4.2099	1.2833	1.1022	0.3991	4.5567	1.9958	1.8238	0.6525	0.3939
M=4	4.4854	1.3055	1.1160	0.4228	4.51573	1.9917	1.8185	0.6820	0.3537
M=5	4.7960	1.3670	1.1852	0.4551	4.5468	2.0628	1.8529	0.7076	0.3682
M=6	4.8352	1.4007	1.2465	0.4774	4.5799	2.0452	1.8922	0.7500	0.3960
M=7	5.0710	1.4944	1.3366	0.5295	4.6567	2.1160	1.9399	0.7685	0.4171
M=8	5.1824	1.5943	1.4527	0.5727	4.7276	2.1354	1.9761	0.7966	0.4434
M=9	5.3562	1.6737	1.5330	0.6198	4.6707	2.1794	2.0345	0.8201	0.4671
M=10	5.5687	1.8151	1.7149	0.7086	4.8581	2.2513	2.1321	0.8863	0.5342
M=20	8.3999	3.6525	3.8353	1.6707	5.6317	2.9482	2.8478	1.4998	1.0674
M=30	12.7867	7.0404	7.7125	3.3744	6.8881	4.0778	3.9782	2.4126	1.9305
M=40	19.3375	11.9614	13.2712	5.8232	8.3053	5.5241	5.3936	3.7325	3.1967
M=50	30.1464	20.1364	22.0022	8.71432	10.6197	7.5664	7.5545	5.5766	4.9728

3.7.2 In control surface using Trapezoidal MFs

If the above experiments are repeated using trapezoidal MFs then it may be also examined that the computational cost of simplified IT2 FLC is higher than the corresponding full IT2 FLC. When M becomes larger, the simplified IT2 FLC becomes faster than the full IT2 FLC. When Gaussian IT2 FSs are used, no matter where the input is, it's firing strengths on the center IT2 FSs are always nonzero intervals, and hence, there are more interval rule firing strengths in the Gaussian FS case than in the trapezoidal FS case.

It is observed the following points from Fig 3.18 and Table 3.13.



(a) (b)

Figure 3.18 Computational cost of the simplified IT2 FLC in control surface computation using trapezoidal IT2 FSs. (a) Total computation time of the 100 control surfaces for different M: the number of IT2 FSs in each input domain. Note that the rulebase has N = M^2 rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of different algorithms to the KM algorithms. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Table 3.13 Computational time using control surface for simplified IT2 FSs when Trapezoidal IT2 FSs are used

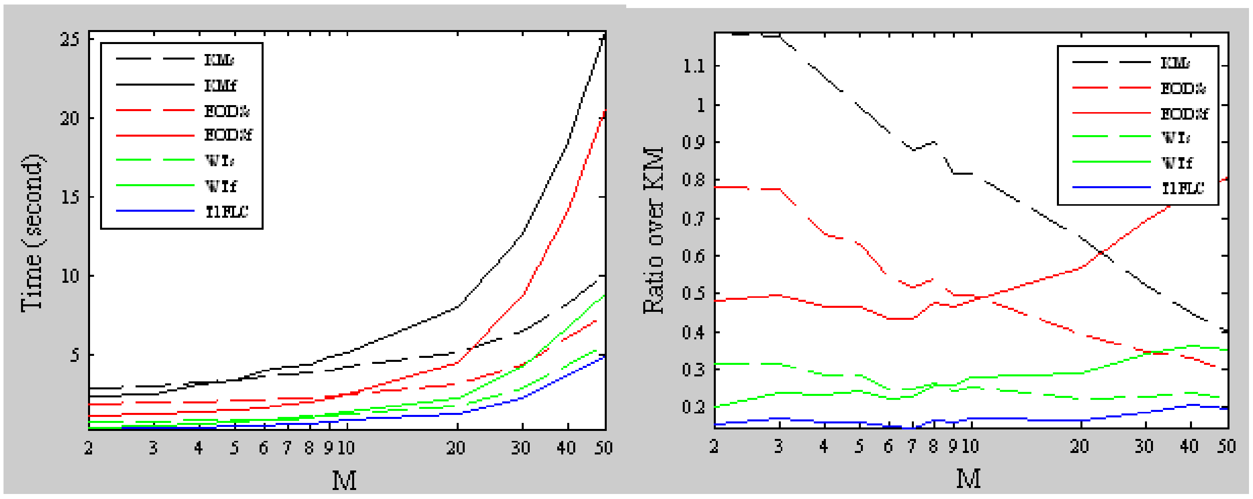
M	Full KM	Full EODS	Full EIASC	Full WT&NT	Simple KM	Simple EODS	Simple EIASC	Simple WT&NT	T1
M=2	2.9375	1.7136	1.6398	0.7694	2.3312	2.1943	2.1107	0.9172	0.4875
M=3	2.9194	1.7437	1.5945	0.7729	2.9651	1.9732	1.8742	0.8029	0.4910
M=4	2.8302	1.7299	1.6037	0.7985	1.6910	1.5513	1.5096	0.6778	0.5044
M=5	3.0453	1.8305	1.6827	0.8532	1.4493	1.3996	1.4100	0.6663	0.5405
M=6	3.1052	2.1644	1.9793	0.9237	1.4696	1.5572	1.4507	0.7500	0.5675
M=7	3.3401	2.0892	1.8617	1.0211	1.5484	1.5128	1.5564	0.7389	0.6283
M=8	3.1624	1.9416	1.8879	0.9893	1.4271	1.5030	1.4725	0.7152	0.6227
M=9	3.2174	2.2296	1.9714	1.0590	1.4237	1.5257	1.5481	0.7465	0.6764
M=10	3.3949	2.2522	2.0782	1.0981	1.6769	1.4322	1.6431	0.8490	0.6940
M=20	5.3538	3.4932	3.6179	2.1137	2.0508	2.0883	2.0456	1.3173	1.3247
M=30	7.0232	5.4369	5.2069	3.6368	2.8829	2.9928	2.9141	2.1354	2.0653
M=40	10.2819	8.0407	8.3560	6.1291	5.1266	5.2125	5.1801	3.5980	3.5606
M=50	16.4703	14.2745	15.0091	13.9812	9.61424	9.1796	8.9812	5.8432	5.6236

1. Simplified KM method is always faster than Full KM method for any value of M.
2. Simplified WT and NT method is faster than Full WT and NT Method when M ≥ 4.
3. All other simplified methods are always faster than its corresponding full methods when M ≥ 4.
4. Among these all methods simplified WT-NT method is the fastest method. But still the computational cost is always less than T1 FLC for M < 40.

3.7.3 In Evolutionary Interval Type-2 Fuzzy Logic Controller using Gaussian MFs

It is also compared the computational cost of the simplified IT2 FLC with the corresponding full IT2 FLC in evolutionary FLC design using Gaussian membership functions. The final results are shown in Fig 3.19 and Table 3.14.The observations are

1. For Gaussian IT2 FSs, the simplified IT2 FLC becomes faster than the corresponding full IT2 FLC when M is large.
2. Among the all simplified and full IT2 FLCs methods simplified WT and NT method is the fastest method.
3. The simplified IT2 FLCs are still slower than T1 FLC for any value of M.
4. Simplified WT-NT method has the less computational cost than its full WT-NT method when $M \geq 9$.
5. Simplified EODS has the less computational cost than its full version when $M \geq 20$.
6. Simplified EIASC method has also the less computational cost than its full EIASC method when $M \geq 20$.
7. Simplified KM method has less computational cost than full KM method when $M > 4$ and the computational cost of simplified version is almost half when $M \geq 20$.



(a) (b)

Figure 3.19 Computational cost of the simplified IT2 FLC in evolutionary FLC design using Gaussian IT2 FSs. (a) Total computation time of the 100 IT2 FLCs for different M: the number of IT2 FSs in each input domain. Note that $N = M^2$. (b) Ratio of the computation time of different algorithms to the KM algorithms. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

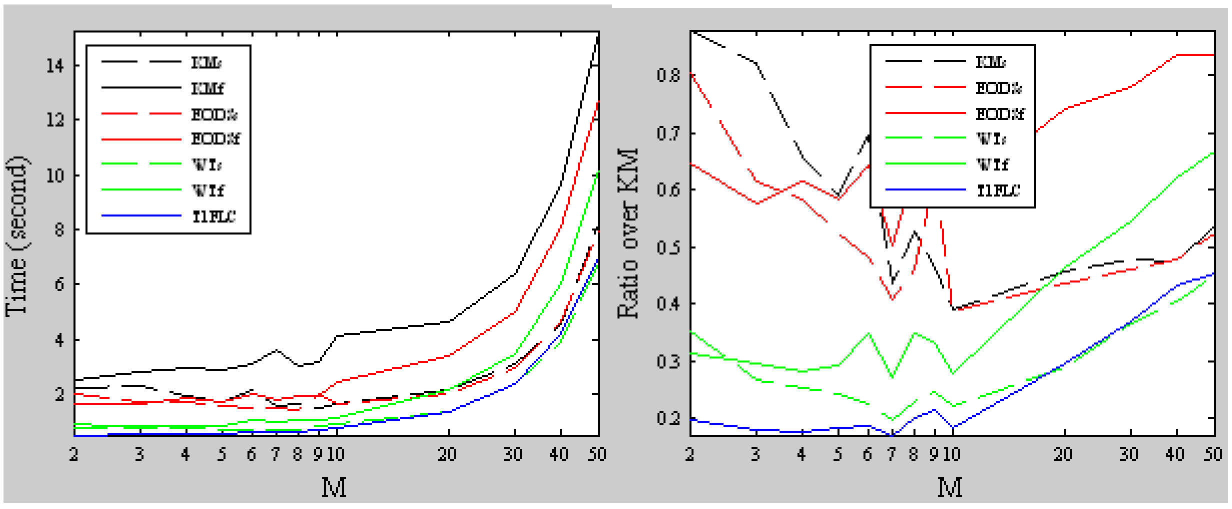
Table 3.14 Computational time using Evolutionary FLC for simplified IT2 FSs when Gaussian IT2 FSs are used

M ($N=M^2$)	Full KM	Full EIASC	Full EODS	Full WT&NT	Simple KM	Simple EIASC	Simple EODS	Simple WT&NT	T1
M=2	2.4429	1.3811	1.1780	0.4868	2.9141	2.0873	1.9114	0.7697	0.3841
M=3	2.6018	1.47432	1.2949	0.6124	3.0647	2.1753	2.0257	0.8197	0.4485
M=4	3.1417	1.5876	1.4567	0.7224	3.3674	2.2182	2.0652	0.8970	0.5046
M=5	3.4108	1.7492	1.5945	0.8212	3.3849	2.3150	2.1519	0.9713	0.5536
M=6	4.0418	1.8813	1.7515	0.9080	3.7450	2.3611	2.2020	0.9934	0.5978
M=7	4.3468	1.9893	1.8840	0.9961	3.8130	2.4089	2.2538	1.0734	0.6394
M=8	4.3779	2.1487	2.0939	1.1311	3.9358	2.5351	2.3623	1.1513	0.7164
M=9	4.9917	2.3581	2.3136	1.2875	4.0742	2.5892	2.4793	1.2202	0.7867
M=10	5.2285	2.4972	2.5046	1.4525	4.2811	2.6619	2.6025	1.3220	0.9007
M=20	8.0860	4.2962	4.5869	2.3279	5.2287	3.3441	3.1943	1.8083	1.3384
M=30	12.6210	7.8607	8.7453	4.2908	6.6133	4.5415	4.3842	2.8852	2.3576
M=40	18.5106	13.0351	14.1596	6.7415	8.3493	6.3394	6.1434	4.4068	3.8385
M=50	25.5577	18.8260	20.5815	8.9782	10.1958	7.8985	7.5807	5.6438	4.9680

3.7.4 In Evolutionary Interval Type-2 Fuzzy Logic Controller using Trapezoidal MFs

The computational cost of the simplified IT2 FLC with the corresponding full IT2 FLC is also compared in evolutionary FLC design using Trapezoidal membership functions. The final results are presented in Fig 3.20 and Table 3.15. It should be notified that

1. For Trapezoidal IT2 FSs, the simplified IT2 FLC becomes faster than the corresponding full IT2 FLC when M is large.



(a) (b)

Figure 3.20 Computational cost of the simplified IT2 FLC in evolutionary FLC design using trapezoidal IT2 FSs. (a) Total computation time of the 100 control surfaces for different M: the number of IT2 FSs in each input domain. Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of different algorithms to the KM algorithms. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

2. Among the all simplified and full IT2 FLCs methods simplified WT and NT method is the fastest method but it is always slower than T1 FLC when M < 20.

3. But the simplified WT-NT method is faster than T1 FLC for large firing rules, basically when $M \geq 20$.

4. Simplified WT-NT method has the less computational cost than its full version for any value of M.

5. Simplified EIASC method is faster than full EIASC method for M > 3.

6. Full EODS method has the higher computational cost than its simplified version when M > 3.

Table 3.15 Computational time using Evolutionary FLC for simplified IT2 FSs when Trapezoidal IT2 FSs are used

M	Full KM	Full EIASC	Full EODS	Full WT&NT	Simple KM	Simple EIASC	Simple EODS	Simple WT&NT	T1
M=2	2.4746	1.6508	1.6024	0.77088	2.1789	2.0458	1.9952	0.8693	0.4875
M=3	2.7820	1.7022	1.6046	0.8173	2.2883	1.7941	1.7143	0.7430	0.4959
M=4	2.9460	1.8037	1.8120	0.8285	1.9380	1.7275	1.7237	0.7472	0.5188
M=5	2.8968	1.7602	1.6884	0.8461	1.7147	1.5606	1.5198	0.6982	0.5336
M=6	3.0741	1.8537	1.9772	1.0707	2.1370	1.9901	1.4767	0.6900	0.5695
M=7	3.5745	2.3634	1.8001	0.9636	1.5636	1.4689	1.4520	0.7062	0.6085
M=8	3.0447	2.3543	1.9353	1.0574	1.6052	1.4593	1.3996	0.6986	0.6140
M=9	3.1938	2.0300	1.9364	1.0576	1.4760	1.7655	1.9959	0.7813	0.6896
M=10	4.0968	2.3916	2.4010	1.1337	1.6015	1.6482	1.5891	0.9127	0.7540
M=20	4.6044	3.4913	3.4177	2.1396	2.1057	2.0222	2.0105	1.3291	1.3523
M=30	6.3980	5.0481	4.9967	3.4745	3.0642	2.9635	2.9384	2.3482	2.3614
M=40	9.6338	8.0031	8.0504	6.0018	4.5655	4.5453	4.5971	3.9052	4.1697
M=50	15.2857	13.1761	12.7720	10.1948	8.2223	7.2459	8.0012	6.8533	6.9357

If the trapezoidal MFs are used then it can be easily seen that the computational cost of all the simplified IT2 FLCs are almost lower than the corresponding full IT2 FLCs. But still now the simplified IT2 FLCs are slower than T1 FLC for any value of M.

In summary, it is demonstrated that a simplified IT2 FLC can save a significant amount of computational cost over a full IT2 FLC, especially when the number of rules ($N=M^2$) is large and the EODS algorithms or the WT (NT) method is used in TR. Mainly the simplified IT2 FLC using the WT (NT) method has the fastest speed.

3.8 Conclusion

IT2 FLSs have proved better abilities to handle uncertainties than their T1 counterparts in several applications. However, their high computational cost may impede them from certain cost-sensitive real world applications. A comprehensive overview and comparison of three categories of methods have been provided to decrease the computational charge of IT2 FLSs. The first category consists of five enhancements to the KM algorithms. Experiments demonstrated that generally they are all faster than the KM algorithms; among them, the EODS algorithms are the fastest for practical IT2 FLSs. Furthermore, the EIASC algorithms, which are much simpler than the EODS algorithms and are at most 1.2 times slower, may also be preferred by professionals for the ease in understanding and implementation.

The second category consists of eleven alternative type-reducers, which have closed-form representation and, hence, are more suitable for analysis. Experiments demonstrated that except for the DY method, all the other ten methods are generally faster than the KM algorithms. Among them, the WT and NT methods are the fastest. Due to high computational cost of DY method is not presented for the comparison in this section. The BMM method may also be preferred because its high stability and robustness have been comprehensively studied.

The third category consists of a simplified structure for IT2 FLCs, which can be merged with any algorithm in the first or second category. It has been demonstrated that a simplified IT2 FLC can save a significant amount of computational charge over a full IT2 FLC, especially when the number of rules ($N=M^2$) is large. Particularly, the simplified IT2 FLC using the WT or NT method has the fastest speed among all other algorithms. However, it is important to note that the first two categories of methods can be applied to all IT2 FLSs, whereas the simplified structure is only designed for control applications. If the IT2 FLS is used in control and the rule base is large, then it may also be worthwhile to use the simplified structure to further save some computational cost.

References

[1] J. M. Mendel and F. Liu, “Super-exponential convergence of the Karnik-Mendel algorithms for computing the centroid of an interval type-2 fuzzy set,” IEEE Transactions on Fuzzy Systems, vol. 15, no. 2, pp. 309–320, April 2007.

[2] J. M. Mendel and H. Wu, “New results about the centroid of a n interval type-2 fuzzy set, including the centroid of a fuzzy granule,” Information Sciences, vol. 177, pp. 360–377, 2007.

[3] J. M. Mendel, “On centroid calculations for type-2 fuzzy sets,” Appl. Comput. Math., vol. 10, no. 1, pp. 88–96, 2011.

[4] H. Hagras, “Type-2 FLCs: A new generation of fuzzy controllers,” IEEE Computat. Intell. Mag., vol. 2, no. 1, pp. 30–43, Feb. 2007.

[5] M. Hsiao, T. H. S. Li, J. Z. Lee, C. H. Chao, and S. H. Tsai, “Design of interval type-2 fuzzy sliding-mode controller,” Inf. Sci., vol. 178, no. 6, pp. 1686–1716, 2008..

[6] D. Wu and J. M. Mendel, “Enhanced Karnik–Mendel algorithms,” IEEE Trans. Fuzzy Syst., vol. 17, no. 4, pp. 923–934, Aug. 2009.

[7] C.-Y. Yeh, W.-H. Jeng, and S.-J. Lee, “An enhanced type-reduction algorithm for type-2 fuzzy sets,” IEEE Trans. Fuzzy Syst., vol. 19, no. 2, pp. 227–240, Apr. 2011.

[8] K. Duran, H. Bernal, and M. Melgarejo, “Improved iterative algorithm for computing the generalized centroid of an interval type-2 fuzzy set,” in Proc. North Amer. Fuzzy Inf. Process. Soc., New York, May 2008, pp. 1–5.

[9] M. Melgarejo, “A fast recursive method to compute the generalized centroid of an interval type-2 fuzzy set,” in Proc. North Amer. Fuzzy Inf. Process. Soc., San Diego, CA, Jun. 2007, pp. 190–194.

[10] D. Wu and M. Nie, “Comparison and practical implementation of type reduction algorithms for type 2 fuzzy sets and systems,” in Proc. IEEE Int. Conf. Fuzzy Syst., Taipei, Taiwan, Jun. 2011, pp. 2131–2138.

[11] H. Z. Hu,G. Zhao, and H.N.Yang, “Fast algorithm to calculate generalized centroid of interval type-2 fuzzy set,” Control Decis., vol. 25, no. 4, pp. 637–640, 2010.

[12] H. Hu, Y.Wang, and Y. Cai, “Advantages of the enhanced opposite direction searching algorithm for computing the centroid of an interval type-2 fuzzy set,” Asian J. Control, vol. 14, no. 6, pp. 1–9, 2012.

[13] C. Li, J. Yi, and D. Zhao, “A novel type-reduction method for interval type-2 fuzzy logic systems,” in Proc. 5th Int. Conf. Fuzzy Syst. Knowl. Discov., vol. 1, Jinan, China, Mar. 2008, pp. 157–161.

[14] S. Coupland and R. I. John, “Geometric type-1 and type-2 fuzzy logic systems,” IEEE Trans. Fuzzy Syst., vol. 15, no. 1, pp. 3–15, Feb. 2007.

[15] M. Nie and W. W. Tan, “Towards an efficient type-reduction method for interval type-2 fuzzy logic systems,” in Proc. IEEE Int. Conf. Fuzzy Syst., Jun. 2008, pp. 1425–1432.

[16] S.-K. Oh, H.-J. Jang, and W. Pedrycz.(2011, Sep.), A comparative experimental study of type-1/ type-2 fuzzy cascade controller based on genetic algorithms and particle swarm optimization. Expert Syst. Appl. [Online]. vol. 38, pp. 11217–11229. http://dx.doi.org/10.1016/j.eswa.2011.02.169.

[17] S. Greenfield, F. Chiclana, S. Coupland, and R. John, “The collapsing method of defuzzification for discretised interval type-2 fuzzy sets,” Inf. Sci., vol. 179, no. 13, pp. 2055–2069, 2008.

[18] C. Li, J. Yi, and D. Zhao, “A novel type-reduction method for interval type-2 fuzzy logic systems,” in Proc. 5th Int. Conf. Fuzzy Syst. Knowl. Discov., vol. 1, Jinan, China, Mar. 2008, pp. 157–161.

[19] C. W. Tao, J. S. Taur, C.-W. Chang, and Y.-H. Chang, "Simplified type-2 fuzzy sliding controller for wing rock system," Fuzzy Sets Syst., 2012, to be published.

[20] M. Nie and W. W. Tan, "Towards an efficient type-reduction method for interval type-2 fuzzy logic systems," in Proc. IEEE Int. Conf. Fuzzy Syst., Jun. 2008, pp. 1425–1432.

[21] D. Wu, "Twelve considerations in choosing between Gaussian and trapezoidal membership functions in interval type-2 fuzzy logic controllers," in Proc. IEEE World Congr. Comput. Intell., Brisbane, Australia, Jun. 2012.

[22] D.Wu and W.W. Tan, "Computationally efficient type-reduction strategies for a type-2 fuzzy logic controller," in Proc. IEEE Int. Conf. Fuzzy Syst., Reno, NV, May 2005, pp. 353–358.

[23] H.Wu and J. M. Mendel, "Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems," IEEE Trans. Fuzzy Syst., vol. 10, no. 5, pp. 622–639, Oct. 2002.

[24] D. Wu and W. W. Tan, "A simplified architecture for type-2 FLSs and its application to nonlinear control," in Proc. IEEE Conf. Cybern. Intell. Syst., Dec. 2004, pp. 485–490.

[25] O. Castillo, P. Melin, A. Alanis, O. Montiel, and R. Sepulveda, "Optimization of interval type-2 fuzzy logic controllers using evolutionary algorithms," Soft Comput., vol. 15, no. 6, pp. 1145–1160, Jun. 2011.

Chapter 4

Conclusions and Future Scopes

The chapter provides a concluding part of the overall thesis. The conclusion and the future direction of the project are enlightened in this chapter. In this conclusion section the Real World Applications of the IT2 FSs are also illustrated.

4.1 Conclusions

The various methods of centroid evaluation of interval type-2 fuzzy sets (IT2 FSs) are explained in this thesis. Then their associated terms such as variance, skewness, fuzziness, cardinality are also been compared. The main idea for type reduction is based on Karnik-Mendel (KM) algorithm. Though there are various techniques for type reduction (T2 FS to T1 FS) in IT2 FS, such as Centroid type reduction, Height type reduction, Center-of-sets type reduction etc. yet Karnik-Mendel algorithm is the most basic approach for Centroid type reduction.

Among various methods for Centroid type reduction, the iterative KM algorithms have high computational cost. It worth a pretty to deploy IT2 FLSs, which may impede them from certain cost-sensitive real world applications. We can categories the centroid type reduction methods in three parts. Here we have described a comprehensive overview and a comparison in order to diminish the computational cost in real world of uncertainties. In first part five enhanced to KM algorithms are illustrated. They all decrease the computational cost over KM algorithm. Hence the computational times for all these methods are minimized. Among these enhanced methods the EODS method is the fastest methods which require less computational charge. EIASC algorithm is much simpler than EODS algorithm, sometimes making it more preferred for understanding and implementation for centroid type reduction in defuzzification process and certain cost reduction applications. But it is 1.2 times slower than EODS algorithm.

In the second categories, we have examined eleven alternative methods. Their closed-form representation makes the analysis more convenient. It is observed that all these methods are faster than KM method with an exceptional DY method. The DY method also has a huge computational cost for which it is not considered in this section for comparison. Among the all alternative TR algorithm WT and NT methods are considered to be the fastest method even faster than enhanced to KM algorithm, EODS methods. These two methods have already decreased the overall computational cost. Here we have also studied the stability and robustness of BMM method for which it can also be preferred for cost reduction applications.

The third category consists of a much simpler structure for IT2 FLCs, which can be merged with any algorithm in the first or second category. Experiments proved that a simplified IT2 FLC can save a major amount of computational cost over a full IT2 FLC, especially when the number of rules is large. Particularly, the simplified IT2 FLC using the WT or NT method has the fastest speed. However, it should be noted that the first two categories of methods can be applied to all IT2 FLSs, whereas the simplified structure is only designed for control applications. Here the third categories are also applied for KM, EIASC, EODS and WT-NT methods. The simplified and full versions of all these algorithms are first constructed then they are compared to each other. It is examined that simplified WT-NT method has the less computational cost over all other methods in three categories.

All the methods depend on the total number of rules are fired. Here the total number of rules is denoted by N, where $N = M^2$. Also here M is represented as the number of Gaussians IT2 FSs in each domain. We consider the value of M for these experiments are M = 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50.

If the sampled value of M is 20 then total 20 × 20 = 400 rules must be fired. Each of the M^2 rule consequents is represented by a crisp number y^n, n = 1, . . . , M^2. The population consisted of 100 randomly generated IT2 FLCs and the performance of each FLC was evaluated by a step response in the first 100 s with sampling frequency 1 Hz. We recorded the time that all the algorithms were used to perform these 100 × 100 = 10000 TRs.

All the comparisons are done here by using control surface and evolutionary interval type-2 fuzzy logic controller. We have used the two types of membership functions such as Gaussians MFs and Trapezoidal MFs to compare all the experiments.

The investigating of this paper will help the researchers who are investigating to find out the correct centroid. There are suitable comprehensive comparisons on the performances of the KM algorithms-based TR approaches and the alternative TR approaches in this paper.

The matlab codes of calculating the centroid, variance, fuzziness, cardinality, skewness are given in this thesis. Also all the matlab codes for both Gaussian MFs and Trapezoidal MFs are provided here. The matlab codes of centroid calculation of a given example using various methods are given in APPENDIX A. The statistics of the absolute difference between the output of the KM TR method and ten alternative TR methods using both Gaussian and Trapezoidal MFs are graphically (by using Matlab Simulation) shown in APPENDIX B.

4.1.1 Real World Applications of IT2 FSs

In this section, some of the IT2FLC real world flourishing applications are illuminated. IT2 FSs have many applications in the field of autonomous mobile robots. As mobile robots trace in real-world chaotic environments must be able to administer under the conditions of imprecision and uncertainty present in such surroundings, IT2FLCs have found good grounds for application in mobile robot control. The interval type-2 FLC was applied to robot control involving indoor and outdoor robots and it was noticed that the IT2FLC always outperformed its type-1 counterpart while using a lesser number of rules. This was particularly detectable through the robot paths and the control surfaces, which graphically represent the unknown function articulated by the FLC.

In [H. Hagras 2004], experiments with robots in outdoor unstructured environments were presented to evaluate the real-time performance of the robot interval type-2 FLC and how it manages the huge amounts of uncertainty and imprecision facing the mobile robots in such changing and dynamic environments. The robots were assessed under various climatic conditions (like rain, wind, sunshine, etc.) and different ground conditions (such as slippery and dry grounds) as well as different times of the day. These experiments also involved the use of different challenging environmental features like metallic and plant edges which present extraneous challenges such as a bad sonar response. It has been shown that the type-2 FLC can better manage uncertainties to give a better response while using a lesser rule base (of 4 rules only). It was also noticed that the type-1 FLC can give a good response under specific weather, ground and robot conditions, but if any of these conditions change (e.g. operating in a windy weather condition), then the type-1 FLC with nine rules will give a bad response as it cannot manage the uncertainties associated with the outdoor environments.

In [J. Figueroa et al. 2005], an IT2FLC was offered for a robotic agent intended to track a mobile object in the context of robot soccer games. In this domain there are many sources of uncertainty which include the image processing algorithms (which could be classified as uncertainty about the FLC inputs) as well as uncertainties in the actuators and networking resources. It was shown that the IT2FLC is intelligent to deal with the involved uncertainty in a better way than the type-1 counterparts. It was also observed that with the IT2FLC, it is not required to include extra rules or fuzzy sets contrary to the T1FLC and as such, the IT2FLC uses a minimum rule base.

In [P. Lin et al. 2005], an IT2FLC was applied to the control of a buck DC-DC converter. DC-DC converters are electronic systems that transform one level of electrical voltage into another level (usually) by a switching action. The DC-DC converters are widely used in personal computers, computer peripherals, and adapters of consumer electronic devices. They are an intriguing subject from a control point of view because of their intrinsic nonlinearity. The control technique for DC-DC converters must handle with their wide input voltage and load variations to ensure stability in any operating condition while providing fast transient response. The control problem is to control the duty cycle so that the output voltage can supply a fixed voltage in the presence of the input voltage uncertainty and load variations. It has been noted that the performance of the IT2FLC is better than its type-1 counterpart where the rise time response of IT2FLC is faster than that of T1FLC with no overshoot in the IT2FLC controlled system [P. Lin et al. 2005].

In [D. Wu and W. Tan 2004], a Genetic Algorithm was used to evolve an IT2FLC to control a liquid-level process. It was tested that both the type-1 and the interval type-2 FLCs are talented to attenuate the oscillations when the modeling uncertainties are small. The liquid level in the tank will eventually reach the desired set-point, though the settling time is curtailed when the IT2FLC is employed. When the modeling uncertainties are larger, the T1FLC will give rise to persistent oscillations while the IT2FLC has the ability to eliminate these oscillations and the liquid level reaches its desired height at a steady state. It was concluded that the IT2FLC is more virile than the T1FLC as the IT2FLC outperforms its type-1 counterpart, especially when the uncertainty is large [D. Wu and W. Tan 2004].

As in the above industrial applications, it was also noticed for the feedback control systems, reported in [R. Sepulveda et al. 2005], that in absence of uncertainties, the T1FLC and the IT2FLC responses are very similar. However, as the level of uncertainties increases, the IT2FLC produces a much improved performance with lower overshoot errors and better settling times than the type-1 counterparts. Thus, it was decided that using an IT2FLC in real-world applications is potentially a better choice as the amount of uncertainty in real world systems is hard to estimate [R. Sepulveda et al. 2005].

In [H. Hagras 2007], an IT2FLC was used for the control of Ambient Intelligent Environments (AIEs). The presented agent architecture used a one-pass (non-iterative) method to learn online the user's particular behaviors and preferences for controlling the AIE in a non-intrusive and seamless manner. The system learns the user behavior by learning his particular rules and type-2 membership functions required by the type-2 fuzzy agent. These can then be adapted incrementally in a life-long learning mode to suit the changing environmental conditions and user preferences. The IT2FLC based agent was evaluated in the Essex intelligent Dormitory (iDorm). Unique experiments were conducted with various users during an extended period (spanning the course of the year) where it was possible to evaluate and demonstrate how the agent can adapt in a life-long learning mode and handle the faced short and long-term uncertainties. The IT2FLC based agents were compared to T1FLC based agents in their ability to model the user's behavior while handling the long-term uncertainties. The results showed that the IT2FLC was able to better model the user behavior and handle the short and long-term uncertainties while using fewer rules.

In [E. Jammeh et al. 2009] an IT2FLC was used to address the congestion control for video streaming across IP networks. In a blocked network, a key problem for video is the fragile nature of the compact stream which means that the loss of particular packets and more generally particular pictures or frames has a knock-on effect at the decoder. Video streaming is a delay-intolerant application implying that it is better to deliver a lower-quality video clip or film than to resend packets. Because of its real-time nature, measurements of available bandwidth are generally performed by observing in real-time the packet arrival statistics of the video stream packets. A network path's available bandwidth is volatile as differing data flows including video streams arrive and leave network links along the path. This uncertainty means that equation-based congestion controllers, especially those that rely on packet loss feedback, are unsuitable. The work reported in [E. Jammeh et al. 2009] has shown that the proposed IT2FLC achieved a better delivered video quality compared to existing traditional controllers and a type-1 FLC. To show the response in different network scenarios, experiments demonstrated the response both in the presence of typical Internet cross-traffic as well as when other video streams occupied a bottleneck on an all-IP network. As All-IP networks are intended for multimedia traffic, it is important to develop a form of congestion control that can transfer to them from the mixed traffic environment of the Internet. It was found that the proposed IT2FLC though specifically designed for Internet conditions can also successfully react to the network conditions of an All-IP network. When the control inputs were subject to noise, the IT2FLC resulted in an order of magnitude performance improvement in comparison to the T1FLC. The IT2FLC also showed reduced packet loss when compared to the other controllers, again resulting in superior delivered video quality. When judged by established criteria such as TCP-friendliness and delayed feedback, fuzzy logic congestion control offers a flexible solution to network bottlenecks.

IT2FLC have been successfully applied to the field of marine diesel engines as reported in [C. Lynch et al. 2005, 2006]. Marine diesel engines are large engines which due to their vast sizes and huge power outputs require accurate and robust speed governing. Accurate speed control of marine diesel engines is of critical importance as important deviations from the speed set point could be baneful in terms of performance and potentially disruptive to the engine. Moreover, for applications such as power generation, the engine speed in rpm must be stable multiples of the generated base frequency i.e. 50Hz frequency would require the engine to operate at 1000 rpm, 1500 rpm etc. Hence, significant speed deviation can cause the generation of incorrect frequencies resulting in loss of synchronization between the generator and associated power grid.

There are several sources of uncertainties facing the speed controller of marine diesel engine including input uncertainties (due to sensor measurements affected by high noise levels i.e.

electromagnetic and radio frequency interference, vibration induced turboelectric cable charges, etc.), output uncertainties (due to actuators wear and tear, etc.) and most significantly uncertainties in operation and load conditions (due to varying loads, weather and sea conditions, wind strength, hull fouling, vessel displacement (dependent on cargo), etc.) as well as designers varying opinions for the choice of the control parameters Due to these uncertainties the current control strategies that employ T1FLCs or PID controllers require continuous tuning to deal with the various faced uncertainties. There have been various developments of this IT2FLC which resulted in commercial embedded Type-2 Neuro FLCs (T2NFC (which employ the iterative Karnik-Mendel procedure and RT2NFC (which employs the Wu-Mendel Uncertainty Bounds method). It was noticed that the performances of the RT2NFC and the T2NFC were similar to the commercially employed Viking 25 PID controller and type-1 FLC when introducing the disturbance of 20 percent load that they were tuned to handle. However, as the uncertainty associated with the change of load increases to 100 percent load. The performance of both the Viking 25 and type-1 FLC degrades significantly, producing large overshoots/undershoots as well as long settling times. The performance of Viking 25 and the type-1 FLC is impermissible under these levels of uncertainties and both controllers are unworkable to real world applications under said conditions. Thus, the common practice in such situations is to retune the controller, which is a time-consuming process. On the other hand, both the RT2NFC and the T2NFC produced IT2FLCs that handled effectively the uncertainties related to the change of the load and operation condition to give a very good performance that has small overshoots/undershoots as well as short settling times. The performance of both the RT2NFC and the T2NFC satisfy the required standards under all conditions and thus require no further tuning. Therefore, the RT2NFC and T2NFC can be used effectively to produce accurate and robust speed controllers for marine diesel engines. This shows the potential of IT2FLCs, which, as the level of uncertainties increases, are able to better manage the uncertainties and outperform the Viking25 and type-1 fuzzy logic controllers.

4.2 Future Scopes

All the experiments mentioned in this paper can be done by using any other membership function. It can further be experimented that using which type of membership function can reduce the overall computational cost for the real world of uncertainties. The DY method which is not considered for alternative type reduction algorithms for its high computational cost than iterative KM algorithm can be modified in future. To identify the embedded T1 FSs for IT2 FSs with arbitrary FOU we should modify the TTCC method in future. Except the Geometric method, the all alternative type reduction algorithms should be modified to $\{y_n\}_{n=1,\ldots,N}$ to be sorted that a better approximation of centroid evaluation is expected to the KM algorithm.

It is observed that in the final expression in equation (2.19), θ_k no longer appears, so that the direct calculus approach does not work. But in future Lagrange's Multiplier can be applied to find out the maximum and minimum value of $y(\theta_1,\ldots,\theta_N)$. The minimum value must be represented as the left centroid (y_l) of an IT2 FS. No other value for x_k should minimum as y_l. The maximun value must be represented as the right centroid (y_r) of an IT2 FS. Also no other value for x_k should maximum as y_r. A better approximation is needed to get the desired output.

It has been verified that a simplified IT2 FLC can save a significant amount of computational cost over a full IT2 FLC, especially when the number of rules ($N=M^2$) is large. So this simplified interval type-2 fuzzy logic systems may be modified in future as they can save significant amount of computational cost when the number of rules is small.

As BMM method has high stability and robustness, so it can be modified in future that it can perform high speed among all other eleven type-reductions algorithms.

References

[1] D. Wu and W. Tan, "Type-2 fuzzy logic controller for the liquid-level process" Proceeding of the 2004 IEEE International Conference on Fuzzy Systems, pp. 248-253, Budapest, Hungary, July 2004

[2] H. Hagras, "A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots," IEEE Transactions on Fuzzy Systems, Vol.12, pp. 524-539, August 2004

[3] H. Hagras, F. Doctor, A. Lopez, A and V. Callaghan, "An Incremental Adaptive Life Long Learning Approach for Type-2 Fuzzy Embedded Agents in Ambient Intelligent Environments," IEEE Transactions on Fuzzy Systems, Vol. 15, No.1, pp. 41-55, February 2007.

[4] H. Hagras, "Type-2 Fuzzy Logic Controllers: Towards a New Approach for Handling Uncertainties in Real World Environments", IEEE Expert Now Tutorial, http://ieeexplore.ieee.org/articleSale/modulesabstract.jsp?mdnumber=EW1084 , June 2008

[5] E. Jammeh, M. Fleury, C. Wagner, H. Hagras, M. Ghanbari, "Interval Type-2 Fuzzy Logic Congestion Control for Video Streaming across IP Networks", to appear in the IEEE Transaction of Fuzzy Systems, December 2009.

[6] C. Lynch, H. Hagras, and V. Callaghan, "Embedded type-2 FLC for real-time speed control of marine and traction diesel engines," Proceeding of the 2005 IEEE International Conference on Fuzzy Systems, pp. 347 –352, Reno, USA, May 2005.

[7] C. Lynch. H. Hagras and V. Callaghan, "Using uncertainty bounds in the design of an embedded real-time type-2 neuro-fuzzy speed Controller for marine diesel engines," Proceeding of the 2006 IEEE International Conference on Fuzzy Systems, pp. 7217-7224, Vancouver, Canada, July 2006.

[8] P. Lin, C. Hsu and T. Lee, "Type-2 fuzzy logic controller design for buck DC-DC converters, " Proceeding of the 2005 IEEE International Conference on Fuzzy Systems, pp. 365-2370, Reno, USA, May 2005.

[9] R. Sepúlveda, O. Castillo, P. Melin, A. Diaz and O. Montiel, “Handling Uncertainty in Controllers Using Type-2 Fuzzy Logic,” Proceeding of the 2005 IEEE International Conference on Fuzzy Systems, pp. 248-253, Reno, USA, July 2005.

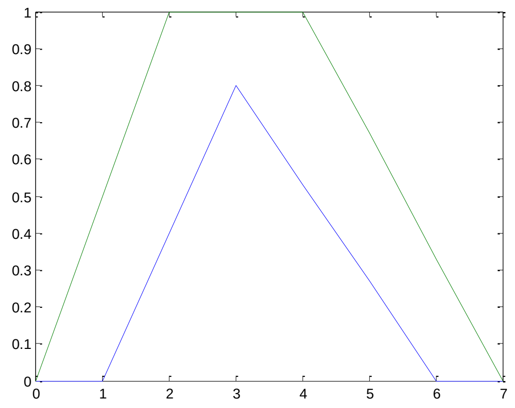
[10] J. Figueroa, J. Posada , J. Soriano, M. Melgarejo and S. Roj, “A type-2 fuzzy logic controller for tracking mobile objects in the context of robotic soccer games,” Proceeding of the 2005 IEEE International Conference on Fuzzy Systems, pp. 359-364, Reno, USA, May 2005.

APPENDIX A

Centroid Evaluation Using Various Algorithms

In this example we consider

```
X=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
```



Matlab Plot of an IT2 FS

KM Algorithm

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];

ly=length(Xl); XrEmpty=isempty(Xr);
if XrEmpty;  Xr=Xl; end
if max(Wl)==0
yl=min(Xl); yr=max(Xr);
y=(yl+yr)/2;  l=1; r=ly-1; return;
end
epsilon=10^(-10);
index=Wr<epsilon;
if sum(index)==ly
yl=min(Xl); yr=max(Xr);
y=(yl+yr)/2; l=1; r=ly-1; return;
end
Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];
ly=length(Xl);
```

```
% Compute yl

    [Xl,index]=sort(Xl); Xr=Xr(index);
    Wl=Wl(index); Wr=Wr(index);
    Wl2=Wl; Wr2=Wr;
if ly==1
    yl=Xl;  l=1;
else
    f = (Wl+Wr)/2 ;
    y0 = Xl*f'/sum(f);
    yl=y0+1; k=1;

    while abs(yl-y0)>epsilon && k<=ly
        yl = y0; k=k+1;
        l = find(Xl > y0,1,'first')-1;
        if l==0; l=1; break; end
        f = [Wr(1:l) Wl(l+1:ly)];
        y0 = Xl*f'/sum(f);
    end
end

% Compute yr

    [Xr,index]=sort(Xr);
    Wl=Wl2(index); Wr=Wr2(index);
if ly==1
    yr=Xr; r=1;
else
    f = (Wl+Wr)/2 ;
    y0 = Xr*f'/sum(f);
    yr=y0+1; k=1;

    while abs(yr-y0)>epsilon && k<=ly
        yr = y0; k=k+1;
        r = find(Xr > y0,1,'first')-1;
        if isempty(r); r=ly-1; break; end
        f = [Wl(1:r) Wr(r+1:ly)];
        y0 = Xr*f'/sum(f);
    end
end
y = (yl+yr)/2
```

EKM Algorithm

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];

ly=length(Xl); XrEmpty=isempty(Xr);
if XrEmpty;  Xr=Xl; end
if max(Wl)==0
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2;  l=1; r=ly-1; return;
end
```

```
epsilon=10^(-10);
index=Wr<epsilon;
if sum(index)==ly
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2; l=1; r=ly-1; return;
end
Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];
ly=length(Xl);

% Compute yl

    [Xl,index]=sort(Xl); Xr=Xr(index);
    Wl=Wl(index); Wr=Wr(index);
    Wl2=Wl; Wr2=Wr;
if ly==1
    yl=Xl;  l=1;
else
    Lold=round(ly/2.4);
    f=[Wr(1:Lold) Wl(Lold+1:ly)];
    a=f*Xl'; b=sum(f); yl=a/b;
    l=find(Xl > yl,1,'first')-1;
    while l~=Lold
        if l==0; l=1; break; end
        minL=min(l,Lold);
        maxL=max(l,Lold);
        delta=sign(l-Lold)*(Wr(minL+1:maxL)-Wl(minL+1:maxL));
        b=b+sum(delta);
        a=a+delta*Xl(minL+1:maxL)';
        yl=a/b;      Lold=l;
        l=find(Xl > yl,1,'first')-1;
    end
end

% Compute yr

    [Xr,index]=sort(Xr);
    Wl=Wl2(index); Wr=Wr2(index);
if ly==1
    yr=Xr; r=1;
else
    Rold=round(ly/1.7);
    f=[Wl(1:Rold) Wr(Rold+1:ly)];

    a=f*Xr'; b=sum(f); yr=a/b;
    r=find(Xr > yr,1,'first')-1;

    while r~=Rold
        minR=min(r,Rold);
        maxR=max(r,Rold);
        delta=sign(r-Rold)*(Wr(minR+1:maxR)-Wl(minR+1:maxR));
        b=b-sum(delta);
        a=a-delta*Xr(minR+1:maxR)';
        yr=a/b;      Rold=r;
        r=find(Xr > yr,1,'first')-1;
        if isempty(r); r=ly-1; break; end
    end
end
y=(yl+yr)/2
```

EKMANI Algorithm

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];

ly=length(Xl); XrEmpty=isempty(Xr);
if XrEmpty;  Xr=Xl; end
if max(Wl)==0
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2;  l=1; r=ly-1; return;
end
epsilon=10^(-10);
index=find(Wr<epsilon);
if length(index)==ly
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2; l=1; r=ly-1; return;
end
Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];
ly=length(Xl);

% Compute yl

    [Xl,index]=sort(Xl); Xr=Xr(index);
    Wl=Wl(index); Wr=Wr(index);
    Wl2=Wl; Wr2=Wr;
if ly==1
    yl=Xl;  l=1;
else
    if nargin >= 6 && ~isempty(l0)
        l0=max(1,min(ly-1,l0));
    else
        l0=round(ly/2.4);
    end
    f=[Wr(1:l0) Wl(l0+1:ly)];
    a=f*Xl'; b=sum(f); yl=a/b;
    l=find(Xl > yl,1,'first')-1;
    while l~=l0 && l~=0
        minL=min(l,l0);  maxL=max(l,l0);
        delta=sign(l-l0)*...
         (Wr(minL+1:maxL)-Wl(minL+1:maxL));
        b=b+sum(delta);
        a=a+delta*Xl(minL+1:maxL)';
        yl=a/b;      l0=l;
        l=find(Xl>yl,1,'first')-1;
    end
end
```

```
% Compute yr

    [Xr,index]=sort(Xr);
    Wl=Wl2(index); Wr=Wr2(index);

if ly==1
    yr=Xr; r=1;
else
    if nargin == 7 && ~isempty(r0);
        r0=max(1,min(ly-1,r0));
    else
        r0=round(ly/1.7);
    end
    f=[Wl(1:r0) Wr(r0+1:ly)];
    a=f*Xr'; b=sum(f); yr=a/b;
    r=find(Xr > yr,1,'first')-1;
    while r~=r0 & ~isempty(r)
        minR=min(r,r0);  maxR=max(r,r0);
        delta=sign(r-r0)*...
        (Wr(minR+1:maxR)-Wl(minR+1:maxR));
        b=b-sum(delta);
        a=a-delta*Xr(minR+1:maxR)';
        yr=a/b;      r0=r;
        r=find(Xr>yr,1,'first')-1;
    end
end
y=(yl+yr)/2
```

IASC Algorithm

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];

ly=length(Xl); XrEmpty=isempty(Xr);
if XrEmpty;  Xr=Xl; end
if max(Wl)==0
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2;  l=1; r=ly-1; return;
end
index=Wr==0;
Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];
ly=length(Xl);

% Compute yl

    [Xl,index]=sort(Xl); Xr=Xr(index);
    Wl=Wl(index); Wr=Wr(index);
    Wl2=Wl; Wr2=Wr;
if ly==1
    yl=Xl;  l=1;
else
    l=0; yl=Xl(end); c=yl;
    a=Xl*Wl'; b=sum(Wl);
    while c <= yl && l < ly
        l=l+1;  yl=c; t=Wr(l)-Wl(l);
        a=a+Xl(l)*t;
        b=b+t;
        c=a/b;
    end
    l=l-1;
end

% Compute yr

    [Xr,index]=sort(Xr);
    Wl=Wl2(index); Wr=Wr2(index);
if ly==1
    yr=Xr; r=1;
else
    r=0;
    a = Xr*Wr';
    b = sum(Wr);
    yr = Xr(1);
    c = yr;
    while c >= yr && r < ly
        r = r+1;
        yr = c; t=Wr(r)-Wl(r);
        a = a-Xr(r)*t;
        b = b-t;
        c = a/b;
    end
    r=r-1;
end
y = (yl+yr)/2
```

EIASC Algorithm

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];

ly=length(Xl); XrEmpty=isempty(Xr);
if XrEmpty;  Xr=Xl; end
if max(Wl)==0
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2;  l=1; r=ly-1; return;
end
index=Wr==0;
Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];
ly=length(Xl);
```

```
% Compute yl

   [Xl,index]=sort(Xl); Xr=Xr(index);
    Wl=Wl(index); Wr=Wr(index);
    Wl2=Wl; Wr2=Wr;
if ly==1
    yl=Xl;  l=1;
else
    yl=Xl(end); l=0;
    a=Xl*Wl'; b=sum(Wl);
    while l<ly && yl > Xl(l+1)
        l=l+1;
        t=Wr(l)-Wl(l);
        a=a+Xl(l)*t;
        b=b+t;
        yl=a/b;
    end
end

% Compute yr

    [Xr,index]=sort(Xr);
    Wl=Wl2(index); Wr=Wr2(index);
if ly==1
    yr=Xr; r=1;
else
    r=ly; yr=Xr(1);
    a=Xr*Wl'; b=sum(Wl);
    while r>0 && yr < Xr(r)
        t=Wr(r)-Wl(r);
        a=a+Xr(r)*t;
        b=b+t;
        yr=a/b;  r=r-1;
    end
end
y=(yl+yr)/2
```

EODS Algorithm

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];

ly=length(Xl); XrEmpty=isempty(Xr);
if XrEmpty;  Xr=Xl; end
if max(Wl)==0
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2;  l=1; r=ly-1; return;
end
index=Wr==0;
Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];
ly=length(Xl);
if ly==0; y=NaN; return; end
```

```
% Compute yl

[Xl,index]=sort(Xl); Xr=Xr(index);
    Wl=Wl(index); Wr=Wr(index);
    Wl2=Wl; Wr2=Wr;
if ly==1
    yl=Xl;  l=1;
elseif ly==2
    l=1; yl=(Xl(1)*Wr(1)+Xl(2)*Wl(2))/(Wr(1)+Wl(2));
else
    m=2; n=ly-1;
    SLm=(Xl(m)-Xl(1))*Wr(1);
    SRn=(Xl(ly)-Xl(n))*Wl(ly);
    Wls=Wl(ly); Wrs=Wr(1);
    while m~=n
        if SLm>SRn
            Wls=Wls+Wl(n); n=n-1;
            SRn=SRn+(Xl(n+1)-Xl(n))*Wls;
        else
            Wrs=Wrs+Wr(m); m=m+1;
            SLm=SLm+(Xl(m)-Xl(m-1))*Wrs;
        end
    end
    if SLm<=SRn
        l=m;
        Wrs=Wrs+Wr(m);
    else
        l=m-1;
        Wls=Wls+Wl(m);
    end
    yl=Xl(m)+(SRn-SLm)/(Wrs+Wls);
end

% Compute yr

    [Xr,index]=sort(Xr);
    Wl=Wl2(index); Wr=Wr2(index);
if ly==1
    yr=Xr; r=1;
elseif ly==2
    r=1; yr=(Xr(1)*Wl(1)+Xr(2)*Wr(2))/(Wl(1)+Wr(2));
else
    m=2; n=ly-1;
    SLm=(Xr(m)-Xr(1))*Wl(1);
    SRn=(Xr(ly)-Xr(n))*Wr(ly);
    Wls=Wl(1); Wrs=Wr(ly);
    while m~=n
        if SLm>SRn
            Wrs=Wrs+Wr(n); n=n-1;
            SRn=SRn+(Xr(n+1)-Xr(n))*Wrs;
        else
            Wls=Wls+Wl(m); m=m+1;
            SLm=SLm+(Xr(m)-Xr(m-1))*Wls;
        end
    end
    if SLm<=SRn
        r=m;
        Wls=Wls+Wl(m);
    else
        r=m-1;
        Wrs=Wrs+Wr(m);
    end
    yr=Xr(m)+(SRn-SLm)/(Wls+Wrs);
end
y = (yl+yr)/2
```

Gorzalczany Method

```
Xl=[0 1 2 3 4 5 6 7]';
Xr=[0 1 2 3 4 5 6 7]';
Wl=[0.0 0.0 0.4 0.8 0.53 0.27 0.0 0.0]';
Wr=[0.0 0.5 1.0 1.0 1.0 0.67 0.33 0.0]';

[X,index]=sort(Xl); Wr=Wr(index); Wl=Wl(index);
uMF=.5*(Wr+Wl).*(1-(Wr-Wl));
x=linspace(X(1),X(end),1001);
mu=uMF(1)*ones(1,length(x));
mu(end)=uMF(end);
for i=2:length(x)-1
    left=find(X<x(i),1,'last');     right=left+1;
    mu(i)=uMF(left)+(uMF(right)-uMF(left))*(x(i)-X(left))/(X(right)-X(left));
end

s=mu(1); half=.5*sum(mu); i=1;
while s<half
    i=i+1; s=s+mu(i);
end
y=x(i);
if abs(s-half)>abs(s-mu(i)-half)
    y=x(i-1);
end
y
```

Coupland–John Geometric Method

```
Xl=[0 1 2 3 4 5 6 7]';
Xr=[0 1 2 3 4 5 6 7]';
Wl=[0.0 0.0 0.4 0.8 0.53 0.27 0.0 0.0]';
Wr=[0.0 0.5 1.0 1.0 1.0 0.67 0.33 0.0]';
[X,index]=sort(Xl); Wr=Wr(index); Wl=Wl(index);
N=length(Wr); points=zeros(2*N,2);
points(:,1)=[Xl(:); Xr(end:-1:1)]; points(:,2)=[Wl; Wr(end:-1:1)];
a=0; b=0;
for i=1:2*N-1
    c=points(i,1)*points(i+1,2)-points(i+1,1)*points(i,2);
    a=a+(points(i,1)+points(i+1,1))*c;
    b=b+c;
end
y=a/(3*b)
```

Wu–Mendel Uncertainty Bound Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
ly=length(Xl); XrEmpty=isempty(Xr);
if XrEmpty;  Xr=Xl; end
if max(Wl)==0
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2 ;return;
end
epsilon=10^(-10);
index=find(Wr<epsilon);
if length(index)==ly
    yl=min(Xl); yr=max(Xr);
    y=(yl+yr)/2; return;
end
Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];

Xl1=min(Xl);
XlM=max(Xl);
yl0=sum(Wl.*Xl)/sum(Wl);
ylM=sum(Wr.*Xr)/sum(Wr);
if isempty(Xr)
    yrM=ylM; yr0=yl0;
    Xr1=Xl1; XrM=XlM;
    Xr=Xl;
else
    Xr1=min(Xr); XrM=max(Xr);
    yrM=sum(Wl.*Xr)/sum(Wl);
    yr0=sum(Wr.*Xr)/sum(Wr);
end
yl_upper=min(yl0,ylM);
yl_lower=yl_upper-(sum(Wr-Wl)/(sum(Wl).*sum(Wr))*Wl.*(Xl-Xl1)...
    .*Wr.*(XlM-Xl)/(Wl.*(Xl-Xl1)+Wr.*(XlM-Xl)));
yl=(yl_lower+yl_upper)/2;

yr_lower=max(yr0,yrM);
yr_upper=yr_lower+(sum(Wr-Wl)/(sum(Wl).*sum(Wr))*Wr.*(Xr-Xr1)...
    .*Wl.*(XrM-Xr)/(Wr.*(Xr-Xr1)+Wl.*(XrM-Xr)));
yr=(yr_lower+yr_upper)/2;

y=(yl+yr)/2
```

Nie–Tan Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
ly=length(Xl); XrEmpty=isempty(Xr);
y=sum(Xl.*(Wl+Wr))/sum(Wl+Wr)
```

BMM Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
ly=length(Xl); XrEmpty=isempty(Xr);
X=Xl;
y=0.5*(sum(Wl.*X)/sum(Wl))+0.5*(sum(Wr.*X)/sum(Wr))
```

LYZ Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
ly=length(Xl); XrEmpty=isempty(Xr);
X=Xl;
yl=min(sum(min((Wl.*X),(Wr.*X)))/sum(Wl),sum(min((Wl.*X),(Wr.*X))/sum(Wr)));
yr=max(sum(max((Wl.*X),(Wr.*X)))/sum(Wl),sum(max((Wl.*X),(Wr.*X))/sum(Wl)));
y=(yl+yr)/2
```

Wu–Tan Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
ly=length(Xl); XrEmpty=isempty(Xr);
X=Xl;
W = Wr-0.5*(Wr-Wl);
y= sum(W.*X)/sum(W)
```

DY Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
N=length(Xl); XrEmpty=isempty(Xr);
X=Xl;
y1=0;

W=(Wl+Wr)/2;
ym=sum(W.*X)/sum(W);
for k=1:2^N
y1 = y1+ym;
end

y=(1/(power(2,N)))*y1
```

LM Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];

fl=sum(Wl);
fr=sum(Wr);
yl=sum(Wl.*Xl)/fl;
yr=sum(Wr.*Xr)/fr;

y=(yl+yr)/2
```

Collapsing Method

```
Xl=[0 1 2 3 4 5 6 7];
Xr=[0 1 2 3 4 5 6 7];
Wl=[0.0,0.0,0.4,0.8,0.53,0.27,0.0,0.0];
Wr=[0.0,0.5,1.0,1.0,1.0,0.67,0.33,0.0];
N=length(Xl);
fn=0;
for k=1:N
    fn=fn+(Wl+Wr)/2;
end
y=sum(Xl.*(fn))/sum(fn)
```

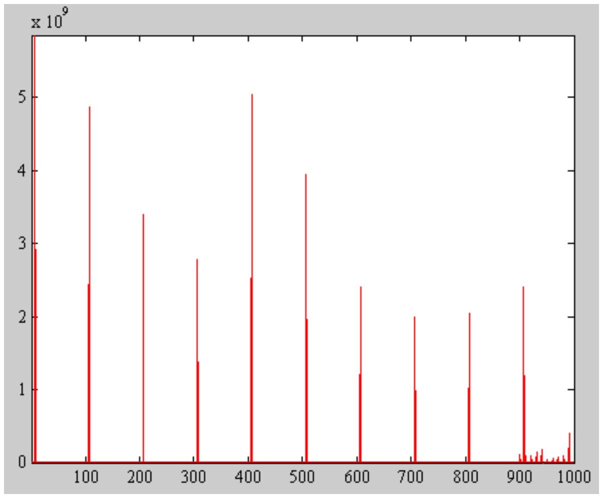
Types of Algorithm	Centroid output
KM	3.3078
EKM	3.3078
EKMANI	3.3078
IASC	3.3078
EIASC	3.3078
EODS	3.3078
Gorzalczany	3.2900
Geometric	3.2640
UB	3.3208
NT	3.3077
BMM	3.3153
LYZ	4.4486
WT	3.3077
DY	3.3077
LM	3.3153
Collapsing	3.3077

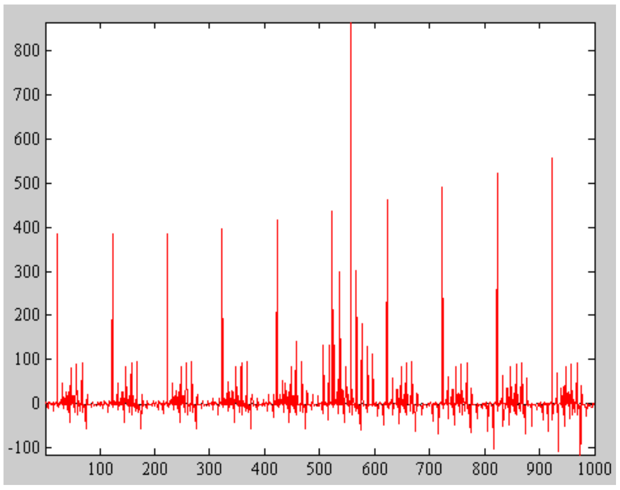
Outputs of different Algorithms

APPENDIX B

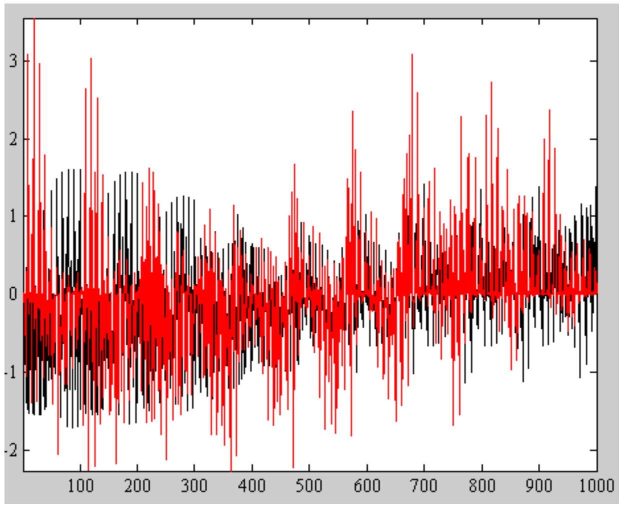
Comparison between the Outputs of the Alternative TR Methods and KM Algorithm using Gaussian MFs and Trapezoidal MFs

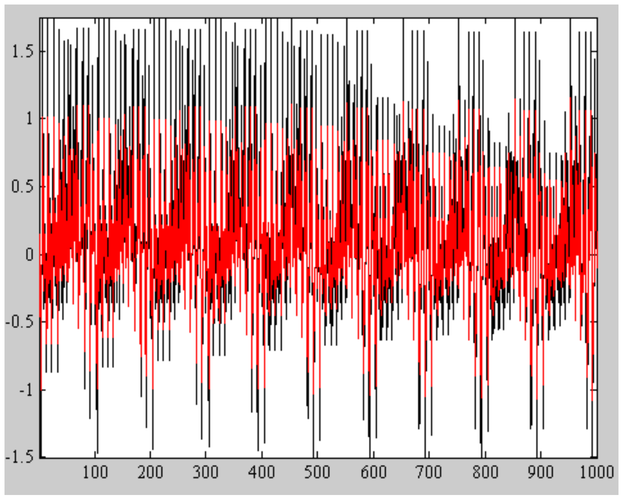
Gaussian MFs **Trapezoidal MFs**





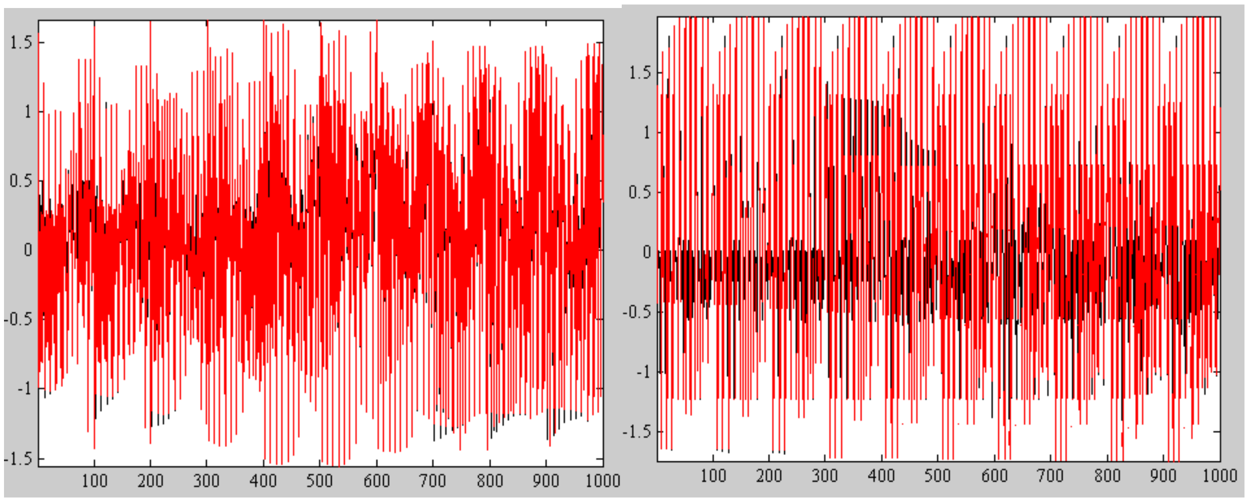
KM Algorithm vs LYZ Method



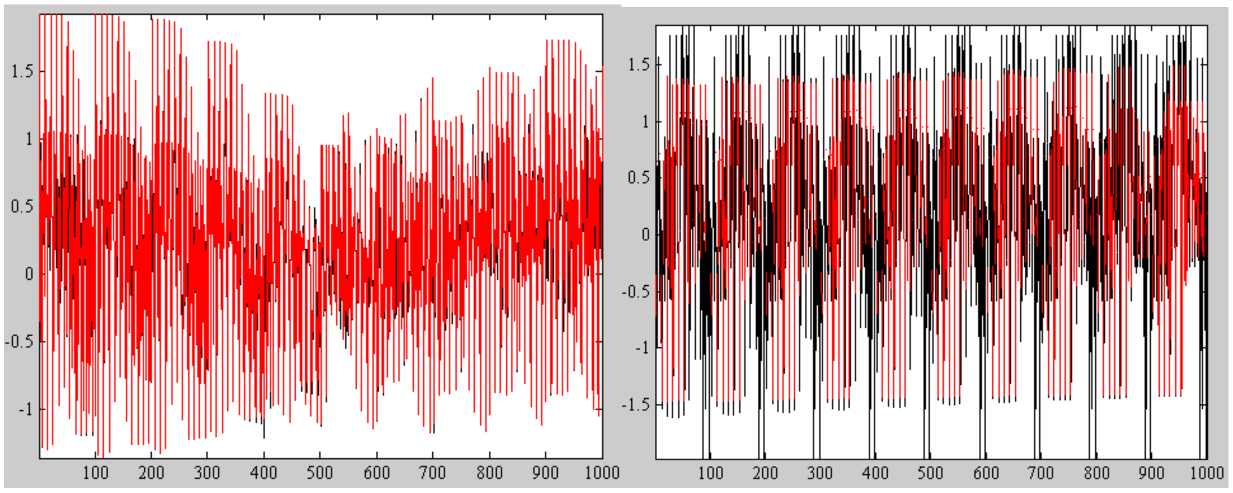


KM Algorithm vs LM Method

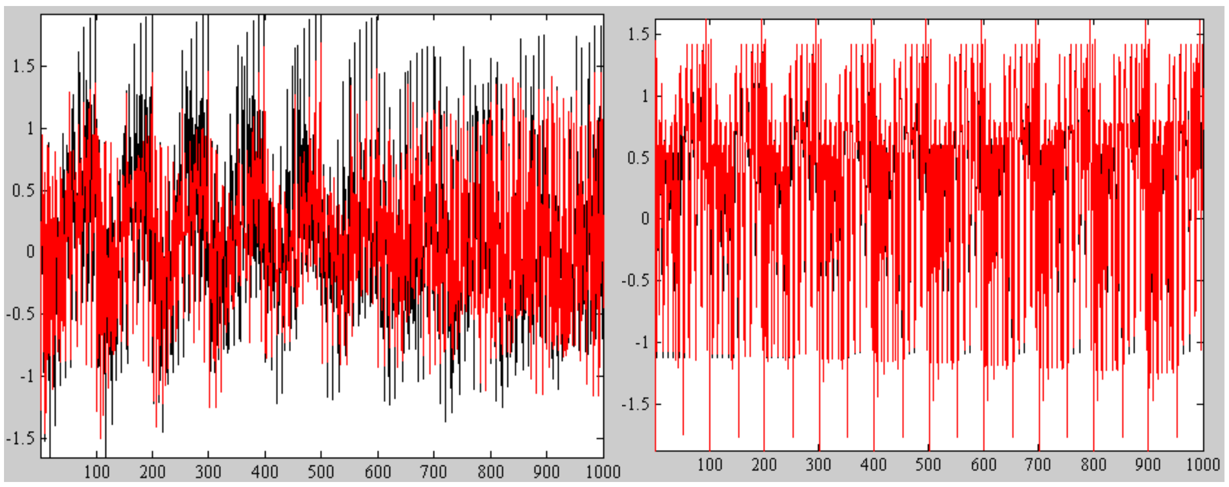
Gaussian MFs **Trapezoidal MFs**



KM Algorithm vs BMM Method

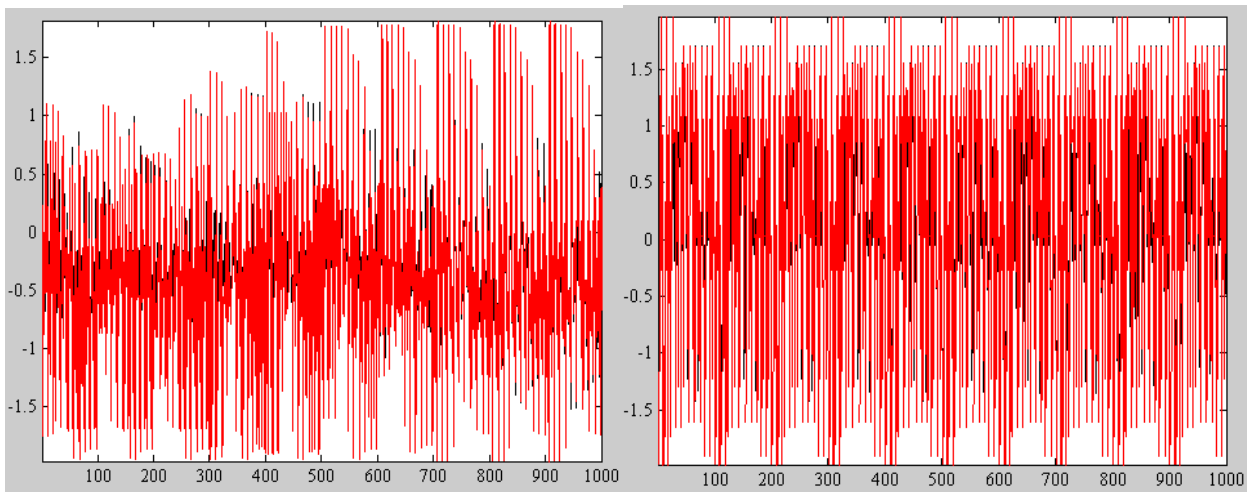


KM Algorithm vs CJG Method

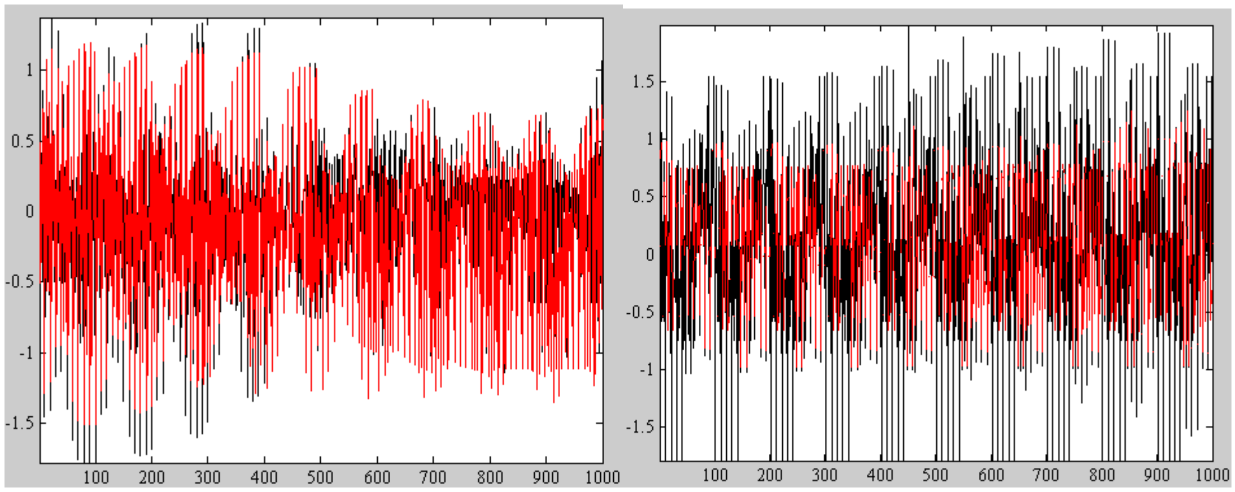


KM Algorithm vs GCCJC Method

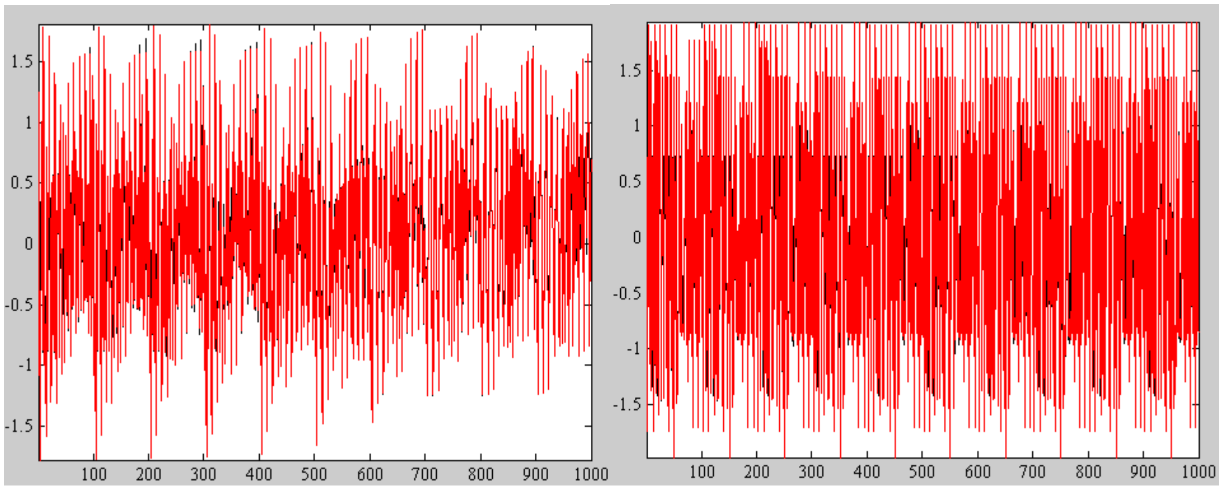
Gaussian MFs **Trapezoidal MFs**



KM Algorithm vs DY Method

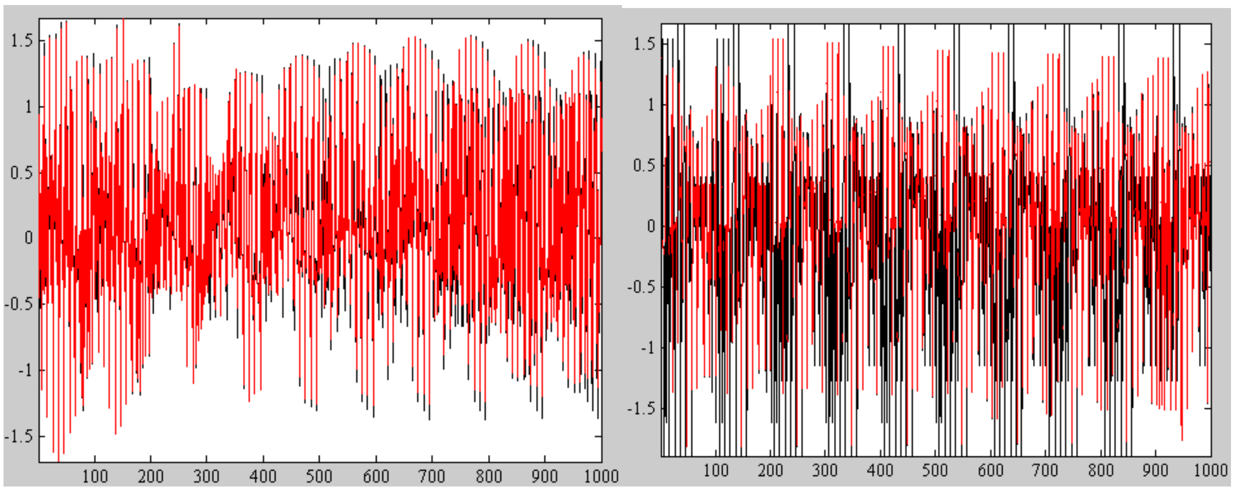


KM Algorithm vs G Method

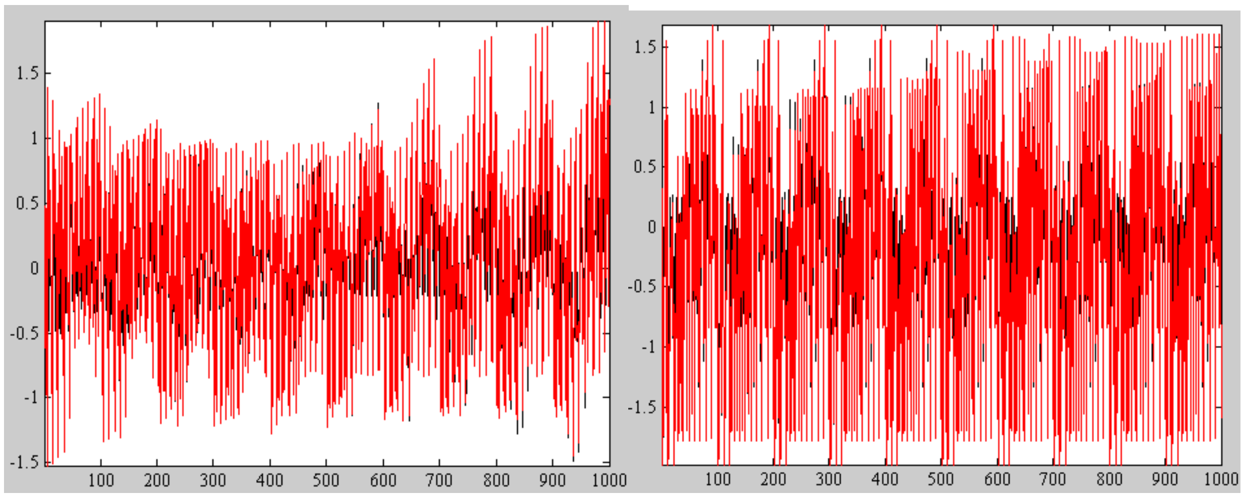


KM Algorithm vs TTCC Method

Gaussian MFs **Trapezoidal MFs**



KM Algorithm vs UB Method



KM Algorithm vs WT&NT Method