Google Search
○ Search WWW [✓ ● Search seattlerobotics.org]

Fuzzy Logic Tutorial

Encoder
The Newsletter of the Seattle Robotics Society

SRS Home | Front Page | Monthly Issue | Index

FUZZY LOGIC - AN INTRODUCTION

PART 1

by Steven D. Kaehler

INTRODUCTION

This is the first in a series of six articles intended to share information and experience in the realm of fuzzy logic (FL) and its application. This article will introduce FL. Through the course of this article series, a simple implementation will be explained in detail. Each article will include additional outside resource references for interested readers.

WHERE DID FUZZY LOGIC COME FROM?

The concept of Fuzzy Logic (FL) was conceived by Lotfi Zadeh, a professor at the University of California at Berkley, and presented not as a control methodology, but as a way of processing data by allowing partial set membership rather than crisp set membership or non-membership. This approach to set theory was not applied to control systems until the 70's due to insufficient small-computer capability prior to that time. Professor Zadeh reasoned that people do not require precise, numerical information input, and yet they are capable of highly adaptive control. If feedback controllers could be programmed to accept noisy, imprecise input, they would be much more effective and perhaps easier to implement. Unfortunately, U.S. manufacturers have not been so quick to embrace this technology while the Europeans and Japanese have been aggressively building real products around it.

WHAT IS FUZZY LOGIC?

In this context, FL is a problem-solving control system methodology that lends itself to implementation in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems. It can be implemented in hardware, software, or a combination of both. FL provides a simple way to arrive at a definite conclusion based upon vague, ambiguous, imprecise, noisy, or missing input information. FL's approach to control problems mimics how a person would make decisions, only much faster.

HOW IS FL DIFFERENT FROM CONVENTIONAL CONTROL METHODS?

FL incorporates a simple, rule-based IF X AND Y THEN Z approach to a solving control problem rather than attempting to model a system mathematically. The FL model is empirically-based, relying on an operator's experience rather than their technical understanding of the system. For example, rather than dealing with temperature control in terms such as "SP =500F", "T <1000F", or "210C <TEMP <220C", terms like "IF (process is too cool) AND (process is getting colder) THEN (add heat to the process)" or "IF (process is too hot) AND (process is heating rapidly) THEN (cool the process quickly)" are used. These terms are imprecise and yet very descriptive of what must actually happen. Consider what you do in the shower if the temperature is too cold: you will make the water comfortable very quickly with little trouble. FL is capable of mimicking this type of behavior but at very high rate.

HOW DOES FL WORK?

FL requires some numerical parameters in order to operate such as what is considered significant error and significant rate-of-change-of-error, but exact values of these numbers are usually not critical unless very responsive performance is required in which case empirical tuning would determine them. For example, a simple temperature control system could use a single temperature feedback sensor whose data is subtracted from the command signal to compute "error" and then time-differentiated to yield the error slope or rate-of-change-of-error, hereafter called "error-dot". Error might have units of degs F and a small error considered to be 2F while a large error is 5F. The "error-dot" might then have units of degs/min with a small error-dot being 5F/min and a large one being 15F/min. These values don't have to be symmetrical and can be "tweaked" once the system is operating in order to optimize performance. Generally, FL is so forgiving that the system will probably work the first time without any tweaking.

SUMMARY

FL was conceived as a better method for sorting and handling data but has proven to be a excellent choice for many control system applications since it mimics human control logic. It can be built into anything from small, hand-held products to large computerized process control systems. It uses an imprecise but very descriptive language to deal with input data more like a human operator. It is very robust and forgiving of operator and data input and often works when first implemented with little or no tuning.

REFERENCES

[1] "Europe Gets into Fuzzy Logic" (Electronics Engineering Times, Nov. 11, 1991).

[2] "Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh", ed. R.R. Yager et al. (John Wiley, New York, 1987).

[3] "U.S. Loses Focus on Fuzzy Logic" (Machine Design, June 21, 1990).

[4] "Why the Japanese are Going in for this 'Fuzzy Logic'" by Emily T. Smith (Business Week, Feb. 20, 1993, pp. 39).

- Back to the Index

  Ahead to Part 2

File: FL_PART1.HTM 2-13-98

FUZZY LOGIC - AN INTRODUCTION

PART 2

by Steven D. Kaehler

INTRODUCTION

This is the second in a series of six articles intended to share information and experience in the realm of fuzzy logic (FL) and its application. This article will continue the introduction with a more detailed look at how one might use FL. A simple implementation will be explained in detail beginning in the next article. Accompanying outside references are included for interested readers.

In the last article, FL was introduced and the thrust of this article series presented. The origin of FL was shared and an introduction to some of the basic concepts of FL was presented. We will now look a little deeper.

WHY USE FL?

FL offers several unique features that make it a particularly good choice for many control problems.

1) It is inherently robust since it does not require precise, noise-free inputs and can be programmed to fail safely if a feedback sensor quits or is destroyed. The output control is a smooth control function despite a wide range of input variations.

2) Since the FL controller processes user-defined rules governing the target control system, it can be

modified and tweaked easily to improve or drastically alter system performance. New sensors can easily be incorporated into the system simply by generating appropriate governing rules.

3) FL is not limited to a few feedback inputs and one or two control outputs, nor is it necessary to measure or compute rate-of-change parameters in order for it to be implemented. Any sensor data that provides some indication of a system's actions and reactions is sufficient. This allows the sensors to be inexpensive and imprecise thus keeping the overall system cost and complexity low.

4) Because of the rule-based operation, any reasonable number of inputs can be processed (1-8 or more) and numerous outputs (1-4 or more) generated, although defining the rulebase quickly becomes complex if too many inputs and outputs are chosen for a single implementation since rules defining their interrelations must also be defined. It would be better to break the control system into smaller chunks and use several smaller FL controllers distributed on the system, each with more limited responsibilities.

5) FL can control nonlinear systems that would be difficult or impossible to model mathematically. This opens doors for control systems that would normally be deemed unfeasible for automation.

HOW IS FL USED?

1) Define the control objectives and criteria: What am I trying to control? What do I have to do to control the system? What kind of response do I need? What are the possible (probable) system failure modes?

2) Determine the input and output relationships and choose a minimum number of variables for input to the FL engine (typically error and rate-of-change-of-error).

3) Using the rule-based structure of FL, break the control problem down into a series of IF X AND Y THEN Z rules that define the desired system output response for given system input conditions. The number and complexity of rules depends on the number of input parameters that are to be processed and the number fuzzy variables associated with each parameter. If possible, use at least one variable and its time derivative. Although it is possible to use a single, instantaneous error parameter without knowing its rate of change, this cripples the system's ability to minimize overshoot for a step inputs.

4) Create FL membership functions that define the meaning (values) of Input/Output terms used in the rules.

5) Create the necessary pre- and post-processing FL routines if implementing in S/W, otherwise program the rules into the FL H/W engine.

6) Test the system, evaluate the results, tune the rules and membership functions, and retest until satisfactory results are obtained.

LINGUISTIC VARIABLES

In 1973, Professor Lotfi Zadeh proposed the concept of linguistic or "fuzzy" variables. Think of them as linguistic objects or words, rather than numbers. The sensor input is a noun, e.g. "temperature", "displacement", "velocity", "flow", "pressure", etc. Since error is just the difference, it can be thought of the same way. The fuzzy variables themselves are adjectives that modify the variable (e.g. "large positive" error, "small positive" error ,"zero" error, "small negative" error, and "large negative" error). As a minimum, one could simply have "positive", "zero", and "negative" variables for each of the parameters. Additional ranges such as "very large" and "very small" could also be added to extend the responsiveness to exceptional or very nonlinear conditions, but aren't necessary in a basic system.

SUMMARY

FL does not require precise inputs, is inherently robust, and can process any reasonable number of inputs but system complexity increases rapidly with more inputs and outputs. Distributed processors would probably be easier to implement. Simple, plain-language IF X AND Y THEN Z rules are used to describe the desired system response in terms of linguistic variables rather than mathematical formulas. The number of these is dependent on the number of inputs, outputs, and the designer's control response goals.

REFERENCES

[5] "Clear Thinking on Fuzzy Logic" by L.A. Bernardinis (Machine Design, April 23, 1993).

[6] "Fuzzy Fundamentals" by E. Cox (IEEE Spectrum, October 1992, pp. 58-61).

[7] "Fuzzy Logic in Control Systems" by C.C. Lee (IEEE Trans. on Systems, Man, and Cybernetics, SMC, Vol. 20, No. 2, 1990, pp. 404-35).

[8] "Fuzzy Sets" by Ivars Peterson (Science News, Vol. 144, July 24, 1993, pp. 55).

- Back to Part 1

  Back to the Index

  Ahead to Part 3

File: FL_PART2.HTM 2-13-98

Encoder
The Newsletter of the Seattle Robotics Society

SRS Home | Front Page | Monthly Issue | Index

FUZZY LOGIC - AN INTRODUCTION

PART 3

by Steven D. Kaehler

INTRODUCTION

This is the third in a series of six articles intended to share information and experience in the realm of fuzzy logic (FL) and its application. This article and the three to follow will take a more detailed look at how FL works by walking through a simple example. Informational references are included at the end of this article for interested readers.

THE RULE MATRIX

In the last article the concept of linguistic variables was presented. The fuzzy parameters of error (command-feedback) and error-dot (rate-of-change-of-error) were modified by the adjectives "negative", "zero", and "positive". To picture this, imagine the simplest practical implementation, a 3-by-3 matrix. The columns represent "negative error", "zero error", and "positive error" inputs from left to right. The rows represent "negative", "zero", and "positive" "error-dot" input from top to bottom. This planar construct is called a rule matrix. It has two input conditions, "error" and "error-dot", and one output response conclusion (at the intersection of each row and column). In this case there are nine possible logical product (AND) output response conclusions.

Although not absolutely necessary, rule matrices usually have an odd number of rows and columns to accommodate a "zero" center row and column region. This may not be needed as long as the functions on either side of the center overlap somewhat and continuous dithering of the output is acceptable since the "zero" regions correspond to "no change" output responses the lack of this region will cause the system to continually hunt for "zero". It is also possible to have a different number of rows than columns. This occurs when numerous degrees of inputs are needed. The maximum number of possible rules is simply the product of the number of rows and columns, but definition of all of these rules may not be necessary since some input conditions may never occur in practical operation. The primary objective of this construct is to map out the universe of possible inputs while keeping the system sufficiently under control.

STARTING THE PROCESS

The first step in implementing FL is to decide exactly what is to be controlled and how. For example, suppose we want to design a simple proportional temperature controller with an electric heating element and a variable-speed cooling fan. A positive signal output calls for 0-100 percent heat while a negative signal output calls for 0-100 percent cooling. Control is achieved through proper balance and control of these two active devices.

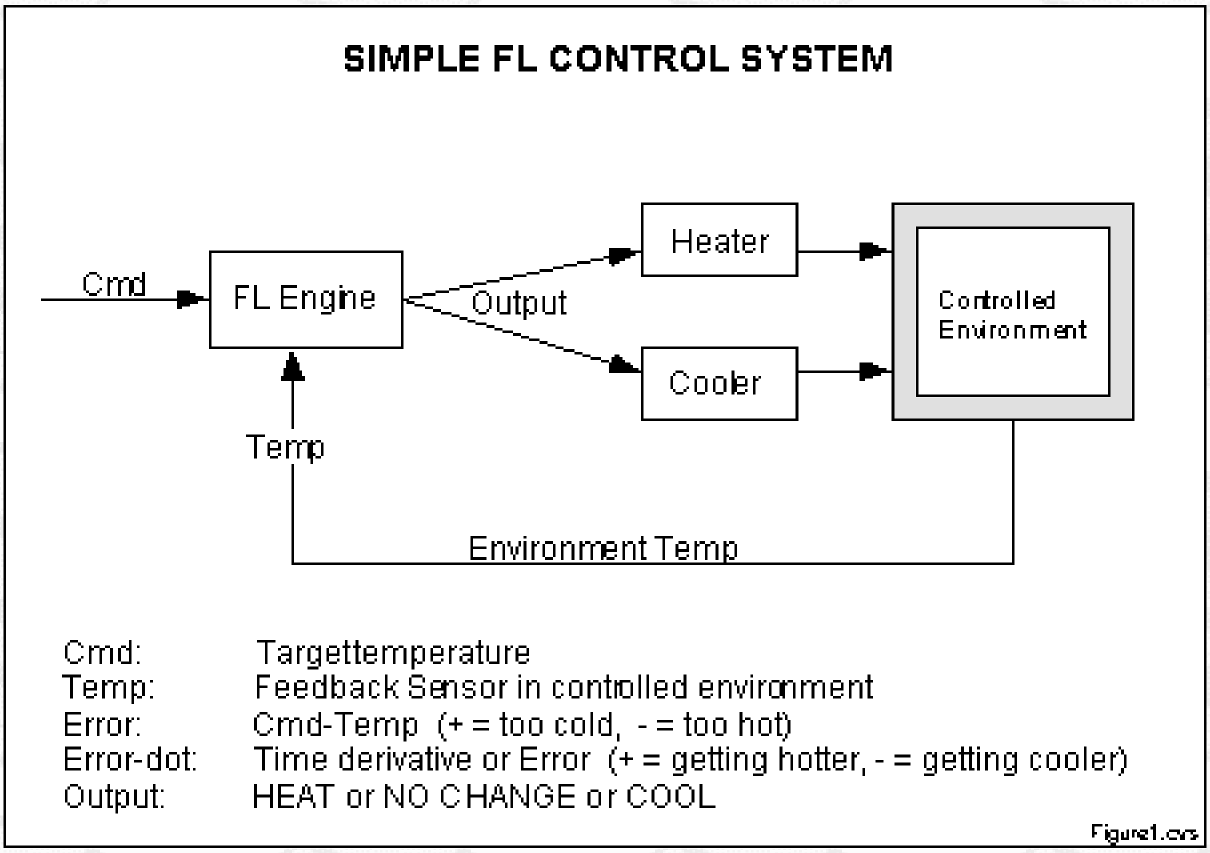

SIMPLE FL CONTROL SYSTEM
Cmd
FL Engine
Output
Heater
Cooler
Controlled Environment
Temp
Environment Temp
Cmd: Targettemperature
Temp: Feedback Sensor in controlled environment
Error: Cmd-Temp (+ = too cold, - = too hot)
Error-dot: Time derivative or Error (+ = getting hotter, - = getting cooler)
Output: HEAT or NO CHANGE or COOL
Figure1.cvs

Figure 1 - A simple block diagram of the control system.

It is necessary to establish a meaningful system for representing the linguistic variables in the matrix. For this example, the following will be used:

"N" = "negative" error or error-dot input level
"Z" = "zero" error or error-dot input level
"P" = "positive" error or error-dot input level
"H" = "Heat" output response
"-" = "No Change" to current output
"C" = "Cool" output response

Define the minimum number of possible input product combinations and corresponding output response conclusions using these terms. For a three-by-three matrix with heating and cooling output responses, all nine rules will need to be defined. The conclusions to the rules with the linguistic variables associated with the output response for each rule are transferred to the matrix.

WHAT IS BEING CONTROLLED AND HOW:

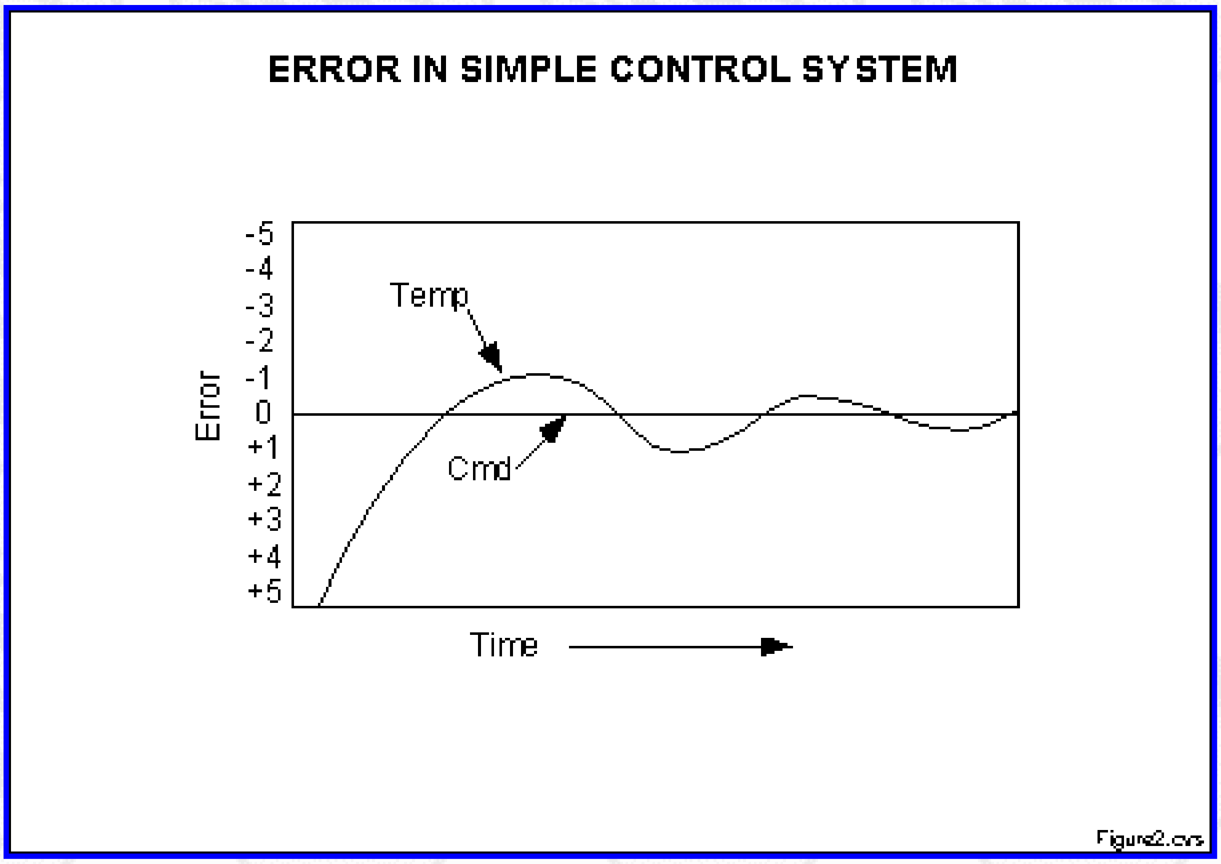


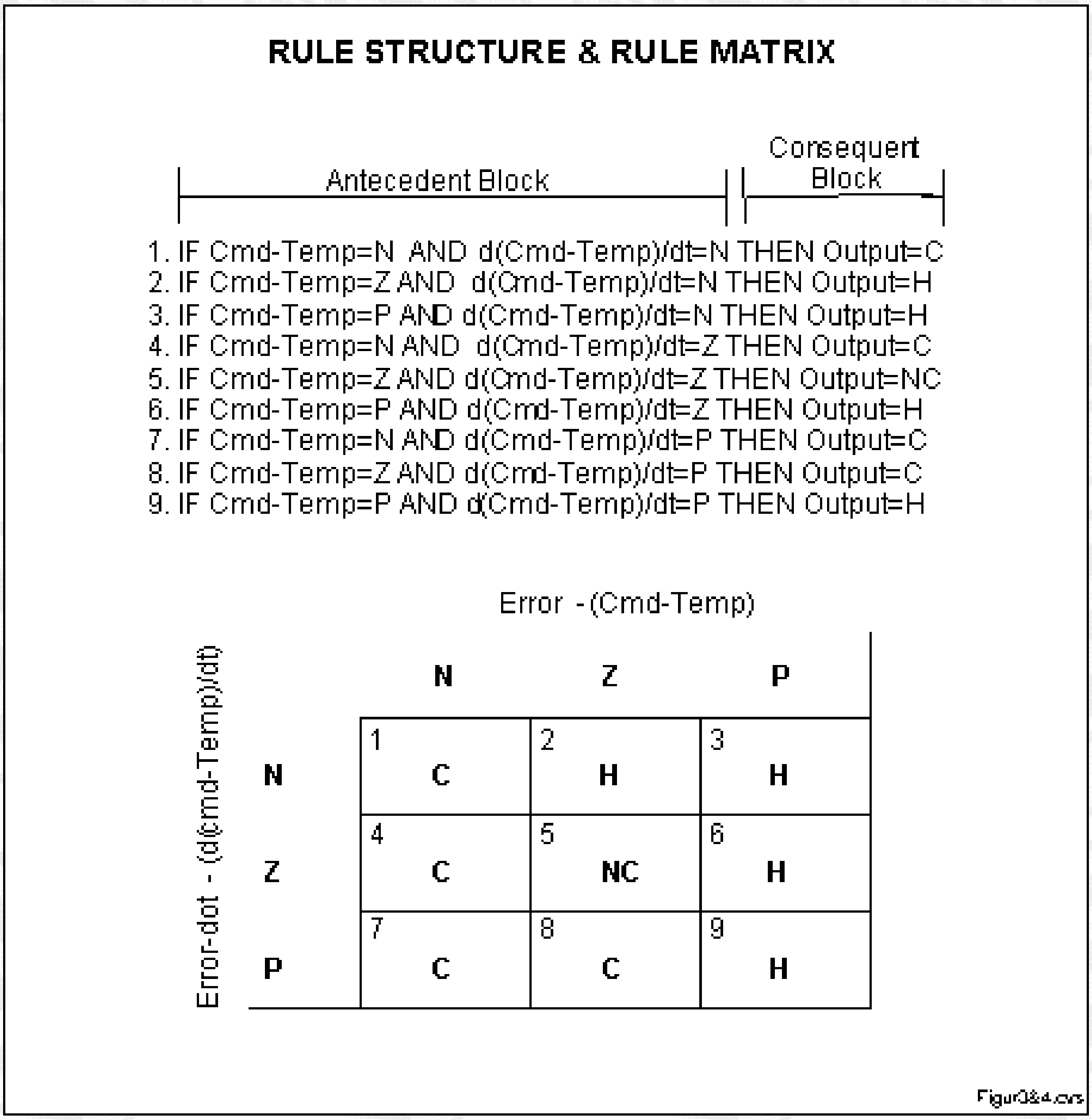
Figure 2 - Typical control system response
Figure 2 shows what command and error look like in a typical control system relative to the command setpoint as the system hunts for stability. Definitions are also shown for this example.

DEFINITIONS:

INPUT#1: ("Error", positive (P), zero (Z), negative (N))
INPUT#2: ("Error-dot", positive (P), zero (Z), negative (N))
CONCLUSION: ("Output", Heat (H), No Change (-), Cool (C))
INPUT#1 System Status
Error = Command-Feedback
P=Too cold, Z=Just right, N=Too hot
INPUT#2 System Status
Error-dot = d(Error)/dt
P=Getting hotter Z=Not changing N=Getting colder
OUTPUT Conclusion & System Response

Output H = Call for heating - = Don't change anything C = Call for cooling
SYSTEM OPERATING RULES

Linguistic rules describing the control system consist of two parts; an antecedent block (between the IF and THEN) and a consequent block (following THEN). Depending on the system, it may not be necessary to evaluate every possible input combination (for 5-by-5 & up matrices) since some may rarely or never occur. By making this type of evaluation, usually done by an experienced operator, fewer rules can be evaluated, thus simplifying the processing logic and perhaps even improving the FL system performance.


Antecedent Block
Consequent Block
Figur3&4.cvs

RULE STRUCTURE & RULE MATRIX

1. IF Cmd-Temp=N AND d(Cmd-Temp)/dt=N THEN Output=C
2. IF Cmd-Temp=Z AND d(Cmd-Temp)/dt=N THEN Output=H
3. IF Cmd-Temp=P AND d(Cmd-Temp)/dt=N THEN Output=H
4. IF Cmd-Temp=N AND d(Cmd-Temp)/dt=Z THEN Output=C
5. IF Cmd-Temp=Z AND d(Cmd-Temp)/dt=Z THEN Output=NC
6. IF Cmd-Temp=P AND d(Cmd-Temp)/dt=Z THEN Output=H
7. IF Cmd-Temp=N AND d(Cmd-Temp)/dt=P THEN Output=C
8. IF Cmd-Temp=Z AND d(Cmd-Temp)/dt=P THEN Output=C
9. IF Cmd-Temp=P AND d(Cmd-Temp)/dt=P THEN Output=H

Error -(Cmd-Temp)

Error-dot - (d(Cmd-Temp)/dt)

	N	Z	P
N	1 C	2 H	3 H
Z	4 C	5 NC	6 H
P	7 C	8 C	9 H

Figures 3 & 4 - The rule structure.
After transferring the conclusions from the nine rules to the matrix there is a noticeable symmetry to the matrix. This suggests (but doesn't guarantee) a reasonably well-behaved (linear) system. This implementation may prove to be too simplistic for some control problems, however it does illustrate the process. Additional degrees of error and error-dot may be included if the desired system response calls for this. This will increase the rulebase size and complexity but may also increase the quality of the control. Figure 4 shows the rule matrix derived from the previous rules.

SUMMARY

Linguistic variables are used to represent an FL system's operating parameters. The rule matrix is a simple graphical tool for mapping the FL control system rules. It accommodates two input variables and expresses their logical product (AND) as one output response variable. To use, define the system using plain-English rules based upon the inputs, decide appropriate output response conclusions, and load these into the rule matrix.

REFERENCES

[9] "Fundamentals of Fuzzy Logic: Parts 1,2,3" by G. Anderson (SENSORS, March-May 1993).

[10] "Fuzzy Logic Flowers in Japan" by D.G. Schartz & G.J. Klir (IEEE Spectrum, July 1992, pp. 32-35).

[11] "Fuzzy Logic Makes Guesswork of Computer Control" by Gail M. Robinson (Design News, Vol. 47, Nov. 28, 1991, pp. 21).

[12] "Fuzzy Logic Outperforms PID Controller" by P. Basehore (PCIM, March 1993).

- Back to Part 2

  Back to the Index

  Ahead to Part 4

File: FL_PART3.HTM 2-13-98

FUZZY LOGIC - AN INTRODUCTION

PART 4

by Steven D. Kaehler

INTRODUCTION

This is the fourth in a series of six articles intended to share information and experience in the realm of fuzzy logic (FL) and its application. This article will continue the example from Part 3 by introducing membership functions and explaining how they work. The next two articles will examine FL inference and defuzzification processes and how they work. For further information, several general references are included at the end of this article.

MEMBERSHIP FUNCTIONS

In the last article, the rule matrix was introduced and used. The next logical question is how to apply the rules. This leads into the next concept, the membership function.

The membership function is a graphical representation of the magnitude of participation of each input. It associates a weighting with each of the inputs that are processed, define functional overlap between inputs, and ultimately determines an output response. The rules use the input membership values as weighting factors to determine their influence on the fuzzy output sets of the final output conclusion. Once the functions are inferred, scaled, and combined, they are defuzzified into a crisp output which drives the system. There are different membership functions associated with each input and output response. Some features to note are:

SHAPE - triangular is common, but bell, trapezoidal, haversine and, exponential have been used. More complex functions are possible but require greater computing overhead to implement.. HEIGHT or magnitude (usually normalized to 1) WIDTH (of the base of function), SHOULDERING (locks height at maximum if an outer function. Shouldered functions evaluate as 1.0 past their center) CENTER points (center of the member function shape) OVERLAP (N&Z, Z&P, typically about 50% of width but can be less).

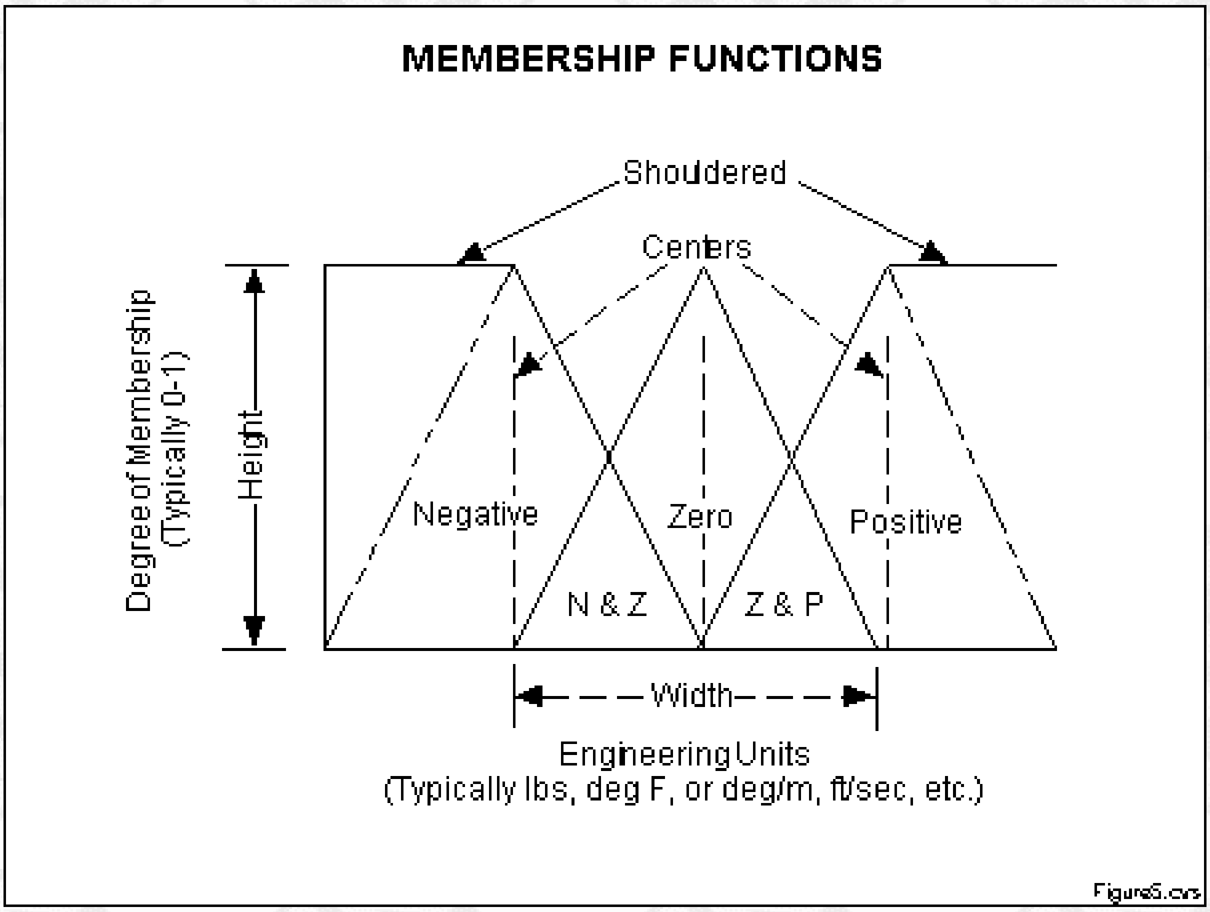


Figure 5 - The features of a membership function

Figure 5 illustrates the features of the triangular membership function which is used in this example because of its mathematical simplicity. Other shapes can be used but the triangular shape lends itself to this illustration.

The degree of membership (DOM) is determined by plugging the selected input parameter (error or error-dot) into the horizontal axis and projecting vertically to the upper boundary of the membership function(s).

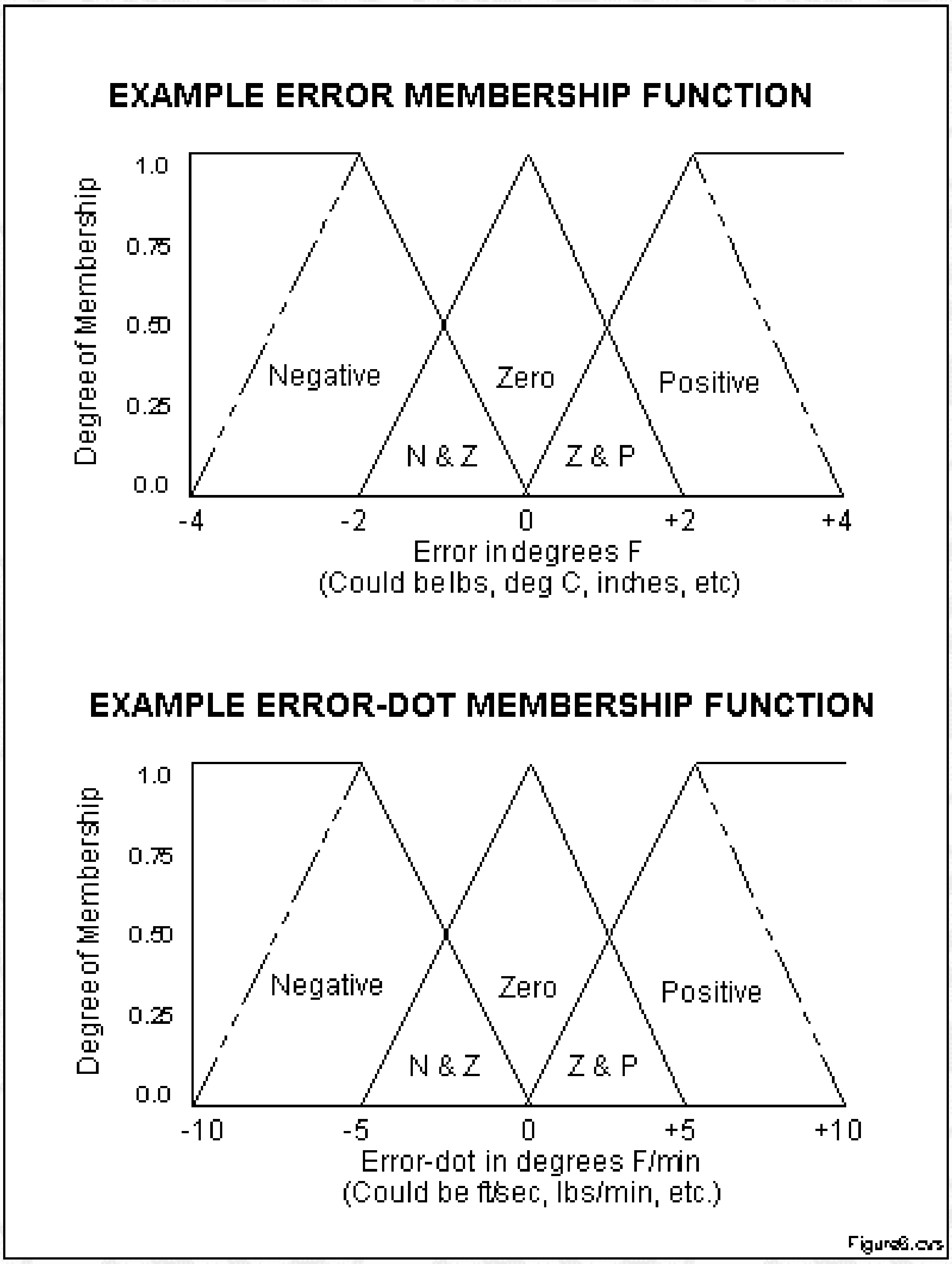


Figure 6 - A sample case

In Figure 6, consider an "error" of -1.0 and an "error-dot" of +2.5. These particular input conditions indicate that the feedback has exceeded the command and is still increasing.

ERROR & ERROR-DOT FUNCTION MEMBERSHIP

The degree of membership for an "error" of -1.0 projects up to the middle of the overlapping part of the "negative" and "zero" function so the result is "negative" membership = 0.5 and "zero" membership = 0.5. Only rules associated with "negative" & "zero" error will actually apply to the output response. This selects only the left and middle columns of the rule matrix.

For an "error-dot" of +2.5, a "zero" and "positive" membership of 0.5 is indicated. This selects the middle and bottom rows of the rule matrix. By overlaying the two regions of the rule matrix, it can be seen that only the rules in the 2-by-2 square in the lower left corner (rules 4,5,7,8) of the rules matrix will generate non-zero output conclusions. The others have a zero weighting due to the logical AND in the rules.

SUMMARY

There is a unique membership function associated with each input parameter. The membership functions associate a weighting factor with values of each input and the effective rules. These weighting factors determine the degree of influence or degree of membership (DOM) each active rule has. By computing the logical product of the membership weights for each active rule, a set of fuzzy output response magnitudes are produced. All that remains is to combine and defuzzify these output responses.

REFERENCES

[13] "Fuzzy but Steady" (1991 Discover Awards) (Discover, Vol. 12, Dec. 1991, pp. 73).

[14] "Neural Networks and Fuzzy Systems--A Dynamic Systems Approach to Machine Intelligence" by B. Kosko (Prentice-Hall, Englewood Cliffs, N.J., 1992).

[15] "Putting Fuzzy Logic into Focus" by Janet J. Barron (Byte, Vol. 18, Apr. 1993, pp. 11).

[16] "Putting Fuzzy Logic in Motion" by Dr. P. Miller (Motion Control, April 1993, pp. 42-44).

- Back to Part 3

  Back to the Index

  Ahead to Part 5

File: FL_PART4.HTM 2-13-98

FUZZY LOGIC - AN INTRODUCTION

PART 5

by Steven D. Kaehler

INTRODUCTION

This is the fifth in a series of six articles intended to share information and experience in the realm of fuzzy logic (FL) and its application. This article will continue the tutorial discussion on FL by looking at the output membership function and several inference processes. The next article will wrap up the discussion of the ongoing example. To further explore the topic of FL, references are included for interested readers.

In the last article, we left off with the inference engine producing fuzzy output response magnitudes for each of the effective rules. These must be processed and combined in some manner to produce a single, crisp (defuzzified) output.

PUTTING IT ALL TOGETHER

As inputs are received by the system, the rulebase is evaluated. The antecedent (IF X AND Y) blocks test the inputs and produce conclusions. The consequent (THEN Z) blocks of some rules are satisfied while others are not. The conclusions are combined to form logical sums. These conclusions feed into the inference process where each response output member function's firing strength (0 to 1) is determined.

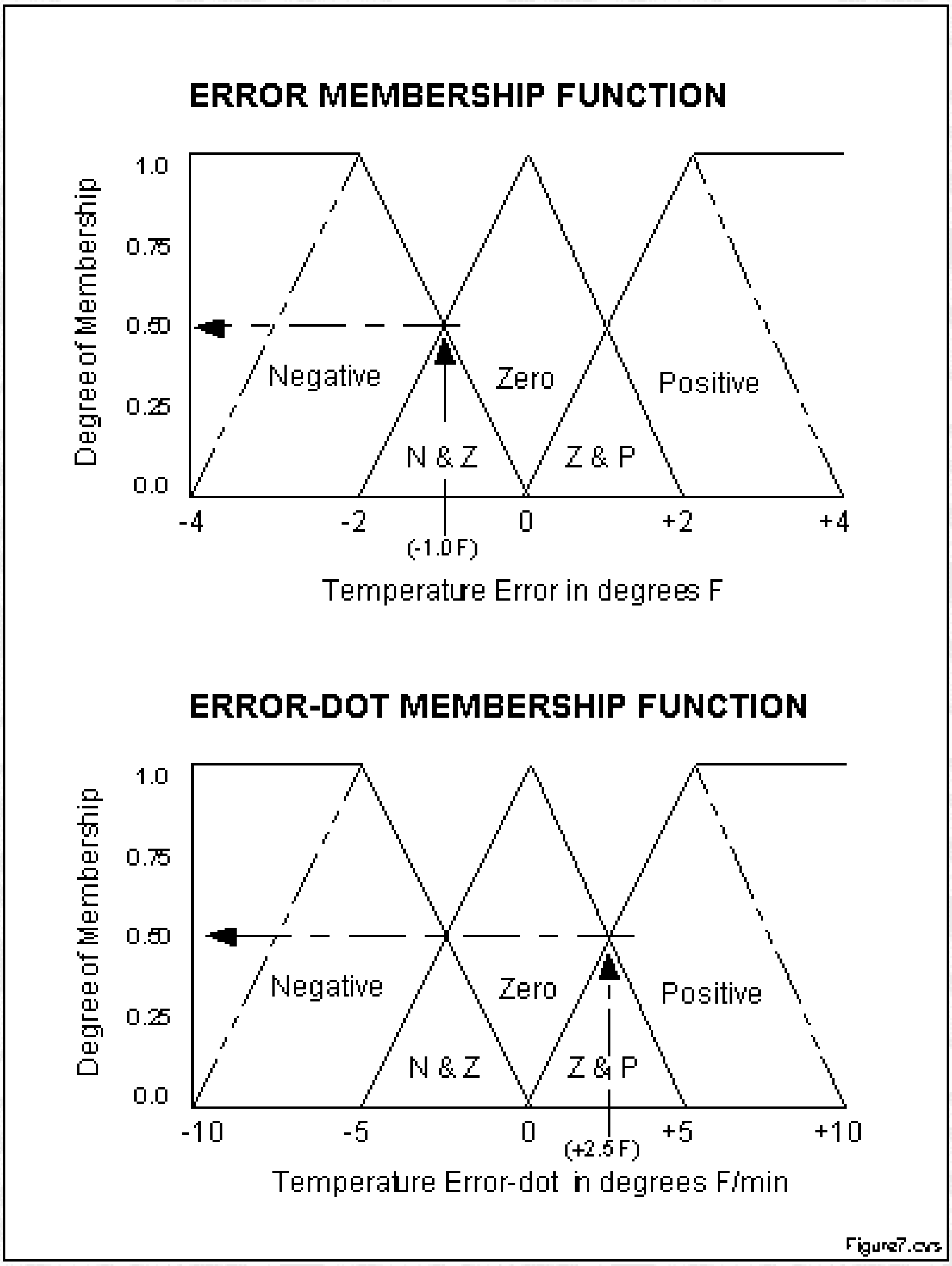


Figure 7 - Degree of membership for the error and error-dot functions in the current example

Data summary from previous illustrations:

INPUT DEGREE OF MEMBERSHIP

"error" = -1.0: "negative" = 0.5 and "zero" = 0.5
"error-dot" = +2.5: "zero" = 0.5 and "positive" = 0.5
ANTECEDENT & CONSEQUENT BLOCKS (e = error, er = error-dot or error-rate)

Now referring back to the rules, plug in the membership function weights from above. "Error" selects rules 1,2,4,5,7,8 while "error-dot" selects rules 4 through 9. "Error" and "error-dot" for all rules are combined to a logical product (LP or AND, that is the minimum of either term).

Of the nine rules selected, only four (rules 4,5,7,8) fire or have non-zero results. This leaves fuzzy output response magnitudes for only "Cooling" and "No_Change" which must be inferred, combined, and defuzzified to return the actual crisp output. In the rule list below, the following ddefinitions apply: (e)=error, (er)=error-dot.

1. If (e < 0) AND (er < 0) then Cool 0.5 & 0.0 = 0.0
2. If (e = 0) AND (er < 0) then Heat 0.5 & 0.0 = 0.0
3. If (e > 0) AND (er < 0) then Heat 0.0 & 0.0 = 0.0
4. If (e < 0) AND (er = 0) then Cool 0.5 & 0.5 = 0.5
5. If (e = 0) AND (er = 0) then No_Chng 0.5 & 0.5 = 0.5
6. If (e > 0) AND (er = 0) then Heat 0.0 & 0.5 = 0.0
7. If (e < 0) AND (er > 0) then Cool 0.5 & 0.5 = 0.5
8. If (e = 0) AND (er > 0) then Cool 0.5 & 0.5 = 0.5
9. If (e > 0) AND (er > 0) then Heat 0.0 & 0.5 = 0.0

SUMMARY

The inputs are combined logically using the AND operator to produce output response values for all expected inputs. The active conclusions are then combined into a logical sum for each membership function. A firing strength for each output membership function is computed. All that remains is to combine these logical sums in a defuzzification process to produce the crisp output.

REFERENCES

[17] "Estimation of Fuzzy Membership from Histograms, Information Sciences" by B.B. Devi et al (Vol. 35, 1985, pp. 43-59).

[18] "Fuzzy Logic" by Bart Kosko and Satoru Isaka (Scientific American, Vol. 269, July 1993, pp. 76).

[19] "Fuzzy Sets, Uncertainty, and Information", by G.J. Klir and T.A. Folger (Prentice-Hall, Englewood Cliffs, N.J., 1988).

[20] "Industrial Applications of Fuzzy Control" ed. M. Sugeno (North-Holland, New York, 1985).

- Back to Part 4

  Back to the Index

  Ahead to Part 6

Encoder
The Newsletter of the Seattle Robotics Society

SRS Home | Front Page |
Monthly Issue | Index

Google Search
○ Search WWW [✓ ● Search seattlerobotics.org]

FUZZY LOGIC - AN INTRODUCTION

PART 6

by Steven D. Kaehler

INTRODUCTION

This is the sixth and final article in a series intended to share information and experience in the realm of fuzzy logic (FL) and its application. This article will conclude the tutorial discussion of the ongoing FL example. For the interested reader, informational references are included.

INFERENCING

The last step completed in the example in the last article was to determine the firing strength of each rule. It turned out that rules 4, 5, 7, and 8 each fired at 50% or 0.5 while rules 1, 2, 3, 6, and 9 did not fire at all (0% or 0.0). The logical products for each rule must be combined or inferred (max-min'd, max-dot'd, averaged, root-sum-squared, etc.) before being passed on to the defuzzification process for crisp output generation. Several inference methods exist.

The MAX-MIN method tests the magnitudes of each rule and selects the highest one. The horizontal coordinate of the "fuzzy centroid" of the area under that function is taken as the output. This method does not combine the effects of all applicable rules but does produce a continuous output function and is easy to implement.

The MAX-DOT or MAX-PRODUCT method scales each member function to fit under its respective peak value and takes the horizontal coordinate of the "fuzzy" centroid of the composite area under the function(s) as the output. Essentially, the member function(s) are shrunk so that their peak equals the magnitude of their respective function ("negative", "zero", and "positive"). This method combines the influence of all active rules and produces a smooth, continuous output.

The AVERAGING method is another approach that works but fails to give increased weighting to more rule votes per output member function. For example, if three "negative" rules fire, but only one "zero" rule does, averaging will not reflect this difference since both averages will equal 0.5. Each function is clipped at the average and the "fuzzy" centroid of the composite area is computed.

The ROOT-SUM-SQUARE (RSS) method combines the effects of all applicable rules, scales the functions at their respective magnitudes, and computes the "fuzzy" centroid of the composite area. This method is more complicated mathematically than other methods, but was selected for this example since it seemed to give the best weighted influence to all firing rules.

DEFUZZIFICATION - GETTING BACK TO CRISP NUMBERS

The RSS method was chosen to include all contributing rules since there are so few member functions associated with the inputs and outputs. For the ongoing example, an error of -1.0 and an error-dot of +2.5 selects regions of the "negative" and "zero" output membership functions. The respective output membership function strengths (range: 0-1) from the possible rules (R1-R9) are:

"negative" = (R1^2 + R4^2 + R7^2 + R8^2) (Cooling) = (0.00^2 + 0.50^2 + 0.50^2 + 0.50^2) ^.5 = 0.866
"zero" = (R5^2)^.5 = (0.50^2)^.5 (No Change) = 0.500
"positive" = (R2^2 + R3^2 + R6^2 + R9^2) (Heating) = (0.00^2 + 0.00^2 + 0.00^2 + 0.00^2) ^.5 = 0.000
A "FUZZY CENTROID" ALGORITHM

The defuzzification of the data into a crisp output is accomplished by combining the results of the inference process and then computing the "fuzzy centroid" of the area. The weighted strengths of each output member function are multiplied by their respective output membership function center points and summed. Finally, this area is divided by the sum of the weighted member function strengths and the result is taken as the crisp output. One feature to note is that since the zero center is at zero, any zero strength will automatically compute to zero. If the center of the zero function happened to be offset from zero (which is likely in a real system where heating and cooling effects are not perfectly equal), then this factor would have an influence.

<u>(neg_center * neg_strength + zero_center * zero_strength + pos_center * pos_strength) </u>= OUTPUT
(neg_strength + zero_strength + pos_strength)

<u>(-100 * 0.866 + 0 * 0.500 + 100 * 0.000) </u>= 63.4%
(0.866 + 0.500 + 0.000)

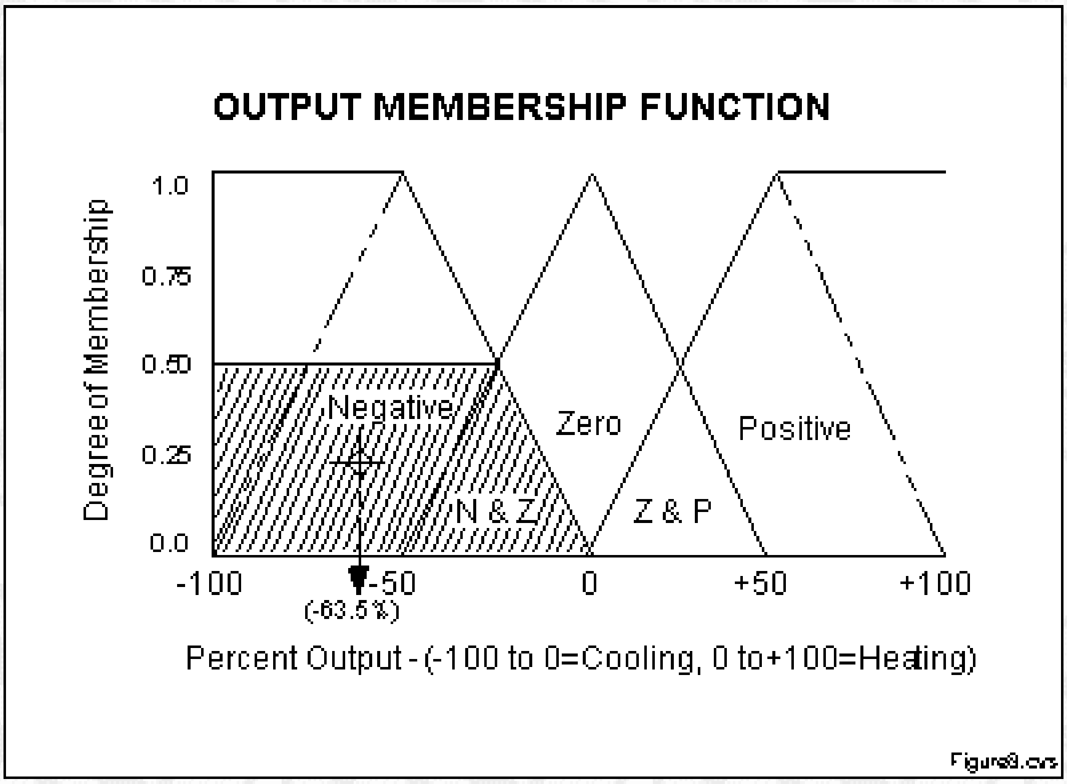


Figure 8 - The horizontal coordinate of the centeriod is taken as the crisp output

The horizontal coordinate of the centroid of the area marked in Figure 8 is taken as the normalized, crisp output. This value of -63.4% (63.4% Cooling) seems logical since the particular input conditions (Error=-1, Error-dot=+2.5) indicate that the feedback has exceeded the command and is still increasing therefore cooling is the expected and required system response.

TUNING AND SYSTEM ENHANCEMENT

Tuning the system can be done by changing the rule antecedents or conclusions, changing the centers of the input and/or output membership functions, or adding additional degrees to the input and/or output functions such as "low", "medium", and "high" levels of "error", "error-dot", and output response. These new levels would generate additional rules and membership functions which would overlap with adjacent functions forming longer "mountain ranges" of functions and responses. The techniques for doing this systematically are a subject unto itself.

SUMMARY

The logical product of each rule is inferred to arrive at a combined magnitude for each output membership function. This can be done by max-min, max-dot, averaging, RSS, or other methods. Once inferred, the magnitudes are mapped into their respective output membership functions, delineating all or part of them. The "fuzzy centroid" of the composite area of the member functions is computed and the final result taken as the crisp output. Tuning the system amounts to "tweaking" the rules and membership function definition parameters to achieve acceptable system response.

CONCLUSION

This completes this article series on FL control and one way it can be done. The author has applied something close to this particular approach to a PC-based temperature controller which could be the topic of a future article series if there is interest. The PC solution has been implemented in Qbasic 1.0 and Borland's Turbo C running on the PC using iotech hardware ADC's and DAC's. This functionality has also been implemented using PIC's and 68HC11 processors.

Fuzzy Logic provides a completely different, unorthodox way to approach a control problem. This method focuses on what the system should do rather than trying to understand how it works. One can concentrate on solving the problem rather than trying to model the system mathematically, if that is even possible. This almost invariably leads to quicker, cheaper solutions. Once understood, this technology is not difficult to apply and the results are usually quite surprising and pleasing.

REFERENCES

[21] "The Coming Age of Fuzzy Logic" Proceedings of the 1989 IFSA Congress, J.C. Bezdek, ed. (University of Washington, Seattle, WA 1989).

[22] "The Current Mode Fuzzy Logic Integrated Circuits Fabricated by Standard CMOS Process" (IEEE Trans. on Computers, Vol. C-35, No. 2, pp. 161-7, February 1986).

[23] "Fuzzy Logic - From Concept to Implementation", (Application Note EDU01V10-0193. ((c) 1993 by Aptronix, Inc, (408) 428-1888).

[24] "Fuzzy Motor Controller" Huntington Technical Brief, D. Brubaker ed. (April 1992, No. 25, Menlo Park, CA 1992).

- Back to Part 5

  Back to the Index

  On to Example Cases

EXAMPLE ILLUSTRATIONS

The following five cases show what the system computes as error decreases toward zero and then changes to a positive value. Pictures of the input and output membership functions are included. The rate-of-change of the error stays constant throughout the five cases. It is not likely that this would happen in a real system, but for purposes of this illustration, that case has been assumed. The values of error and error-dot indicated from the membership functions are plugged into the rulebase from the "KEY" below and the responses computed for each case. These responses are then mathematically combined to yield a crisp output.

Note that because the "zero" membership function is centered on zero in the output function, its influence in the output computation is only in the denominator. The center of the "zero" doesn't need to be at zero, it just happens to be in this example..

KEY:

(e<0) "negative" error value (er<0) "negative" error-dot value
(e=0) "zero" error value (er=0) "zero" error-dot value
(e>0) "positive" error value (er>0) "positive" error-dot value
EXAMPLE CASE 1 - ERROR= -1.0F

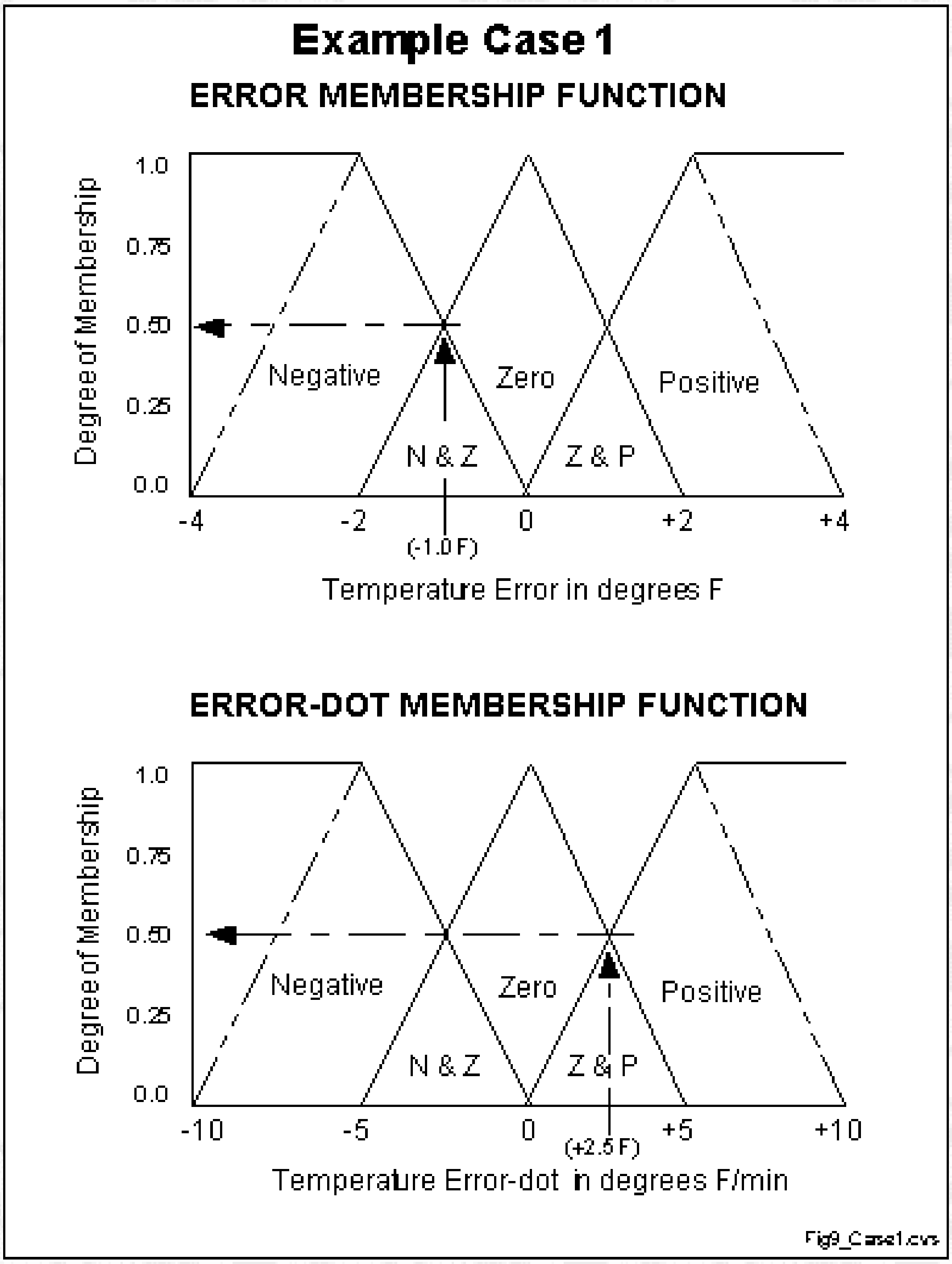


Figure 9 - EXAMPLE CASE 1: (initial conditions)

INPUT DEGREE OF MEMBERSHIP

"error" = -1.0: "negative" = 0.5, "zero" = 0.5, "positive" = 0.0
"error-dot" = +2.5: "negative" = 0.0, "zero" = 0.5, "positive" = 0.5

1. If (e < 0) AND (er < 0) then Cool 0.50 & 0.00 = 0.00
2. If (e = 0) AND (er < 0) then Heat 0.50 & 0.00 = 0.00
3. If (e > 0) AND (er < 0) then Heat 0.00 & 0.00 = 0.00 >>>>
4. If (e < 0) AND (er = 0) then Cool 0.50 & 0.50 = 0.50 >>>>
5. If (e = 0) AND (er = 0) then No_Chng 0.50 & 0.50 = 0.50
6. If (e > 0) AND (er = 0) then Heat 0.00 & 0.50 = 0.00 >>>>

7. If (e < 0) AND (er > 0) then Cool 0.50 & 0.50 = 0.50 >>>>
8. If (e = 0) AND (er > 0) then Cool 0.50 & 0.50 = 0.50
9. If (e > 0) AND (er > 0) then Heat 0.00 & 0.50 = 0.00

"negative" = (R1^2 + R4^2 + R7^2 + R8^2) (Cooling) = (0.00^2 + 0.50^2 + 0.50^2 + 0.50^2) ^.5 = 0.866
"zero" = (R5^2)^.5 = (0.50^2)^.5 (No Change) = 0.500
"positive" = (R2^2 + R3^2 + R6^2 + R9^2) (Heating) = (0.00^2 + 0.00^2 + 0.00^2 + 0.00^2) ^.5 = 0.000

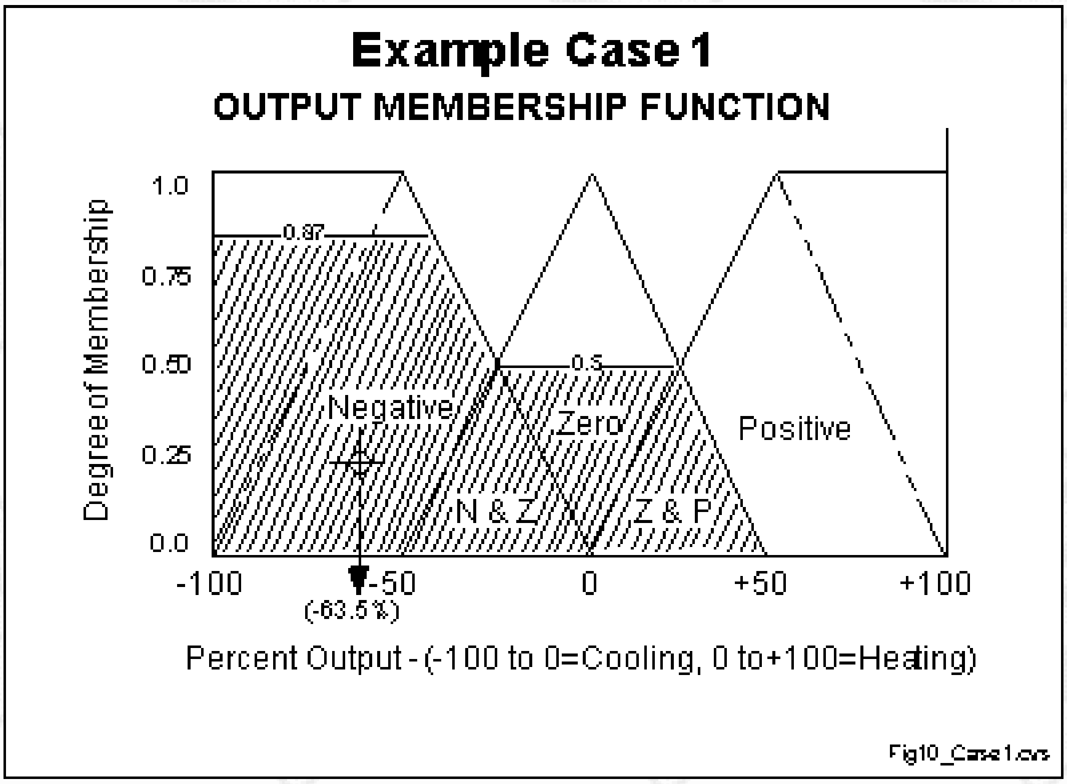


Figure 10 - Output function

<u>(neg_center * neg_strength + zero_center * zero_strength + pos_center * pos_strength)</u> = OUTPUT
(neg_strength + zero_strength + pos_strength)

<u>(-100 * 0.866 + 0 * 0.500 + 100 * 0.000)</u> = -63.4% (cooling)
(0.866 + 0.500 + 0.000)
EXAMPLE CASE 2 - ERROR = +1.25F

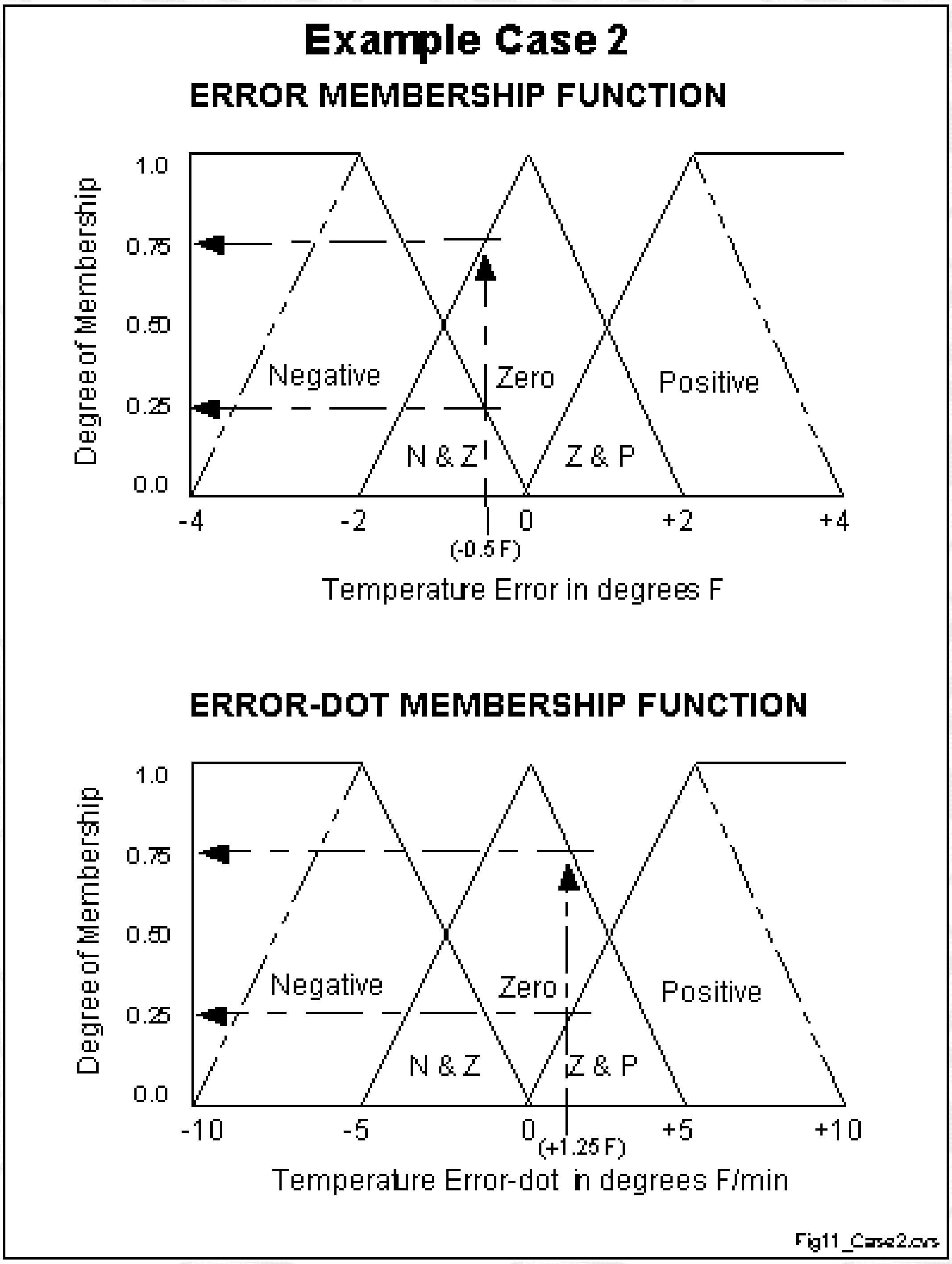


Figure 11 - EXAMPLE CASE 2

INPUT DEGREE OF MEMBERSHIP

"error" = -0.5: "negative" = 0.25, "zero" = 0.75, "positive" = 0.0
"error-dot" = +2.5: "negative" = 0.0, "zero" = 0.50, "positive" = 0.50

1. If (e < 0) AND (er < 0) then Cool 0.25 & 0.0 = 0.00
2. If (e = 0) AND (er < 0) then Heat 0.75 & 0.0 = 0.00
3. If (e > 0) AND (er < 0) then Heat 0.00 & 0.0 = 0.00 >>>>
4. If (e < 0) AND (er = 0) then Cool 0.25 & 0.50 = 0.25 >>>>
5. If (e = 0) AND (er = 0) then No_Chng 0.75 & 0.50 = 0.50
6. If (e > 0) AND (er = 0) then Heat 0.00 & 0.50 = 0.00 >>>>

7. If (e < 0) AND (er > 0) then Cool 0.25 & 0.50 = 0.25 >>>>
8. If (e = 0) AND (er > 0) then Cool 0.75 & 0.50 = 0.50
9. If (e > 0) AND (er > 0) then Heat 0.00 & 0.50 = 0.00

"negative" = (R1^2 + R4^2 + R7^2 + R8^2) (Cooling) = (0.00^2 + 0.25^2 + 0.25^2 + 0.50^2)^.5 = 0.612
"zero" = (R5^2)^.5 = (0.50^2)^.5 (No Change) = 0.50
"positive" = (R2^2 + R3^2 + R6^2 + R9^2) (Heating) = (0.00^2 + 0.00^2 + 0.00^2 + 0.00^2)^.5 = 0.000

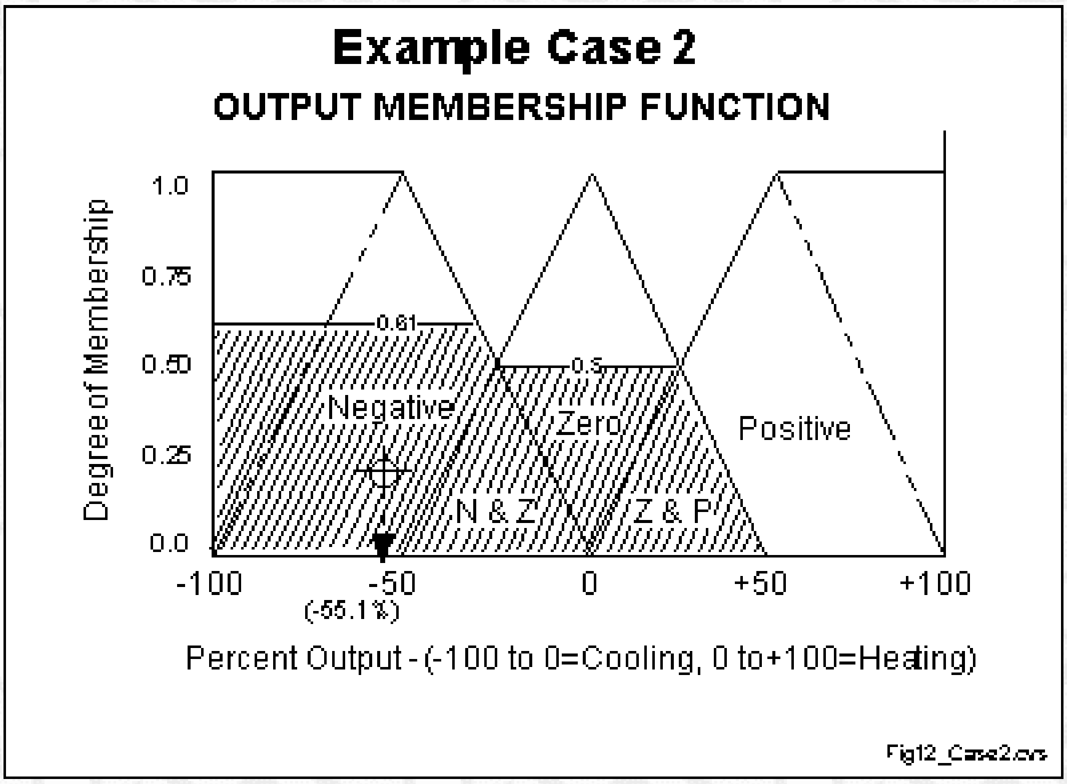


Figure 12 - Output function

(neg_center * neg_strength + zero_center * zero_strength + pos_center * pos_strength) = OUTPUT
(neg_strength + zero_strength + pos_strength)

(-100 * 0.612 + 0 * 0.50 + 100 * 0.000) = -55.1% (cooling)
(0.612 + 0.500 + 0.000)
EXAMPLE CASE 3 - ERROR = 0.0F

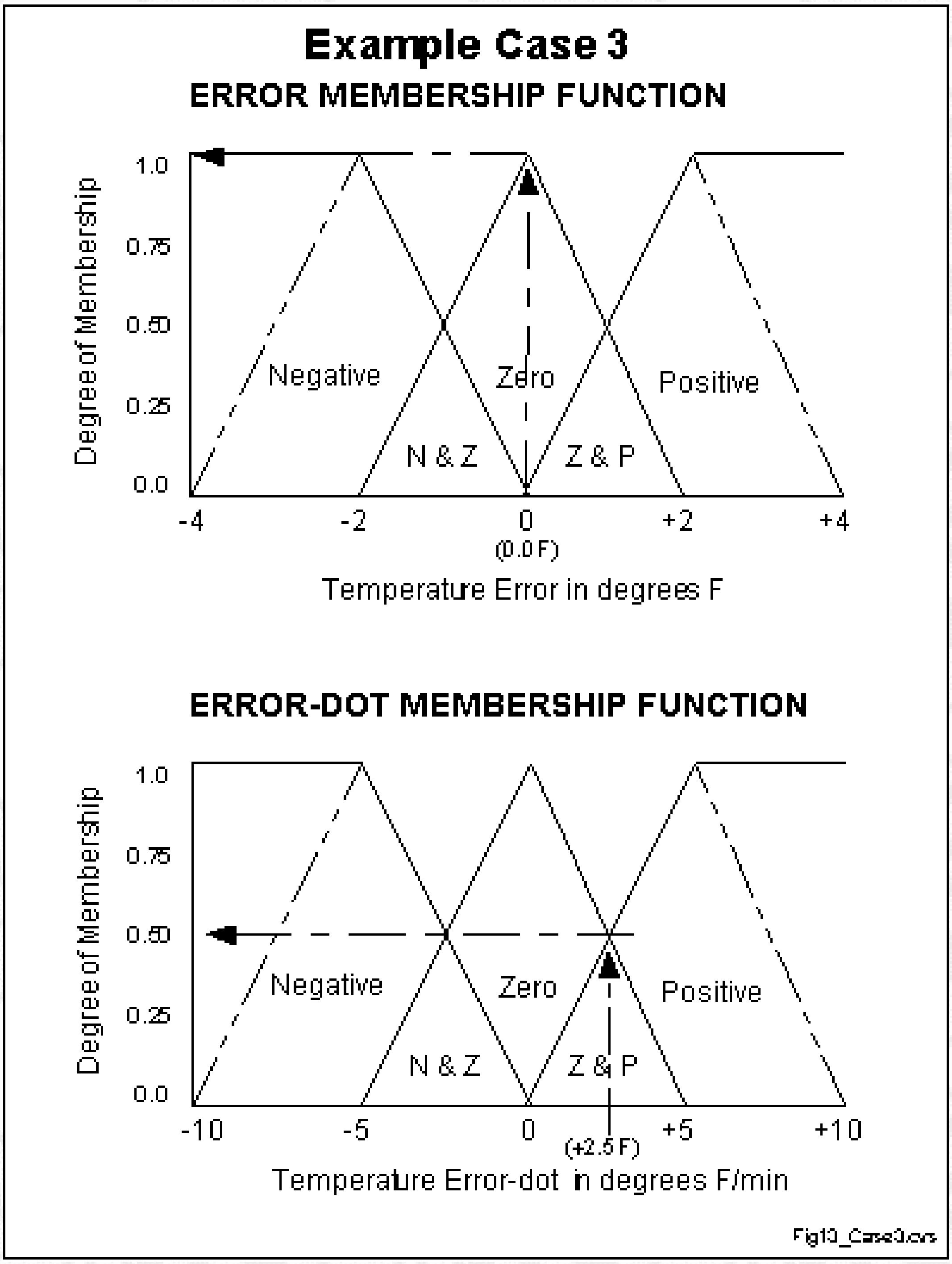


Figure 13 - EXAMPLE CASE 3

INPUT DEGREE OF MEMBERSHIP

"error" = 0.0: "negative" = 0.0, "zero" = 1.0, "positive" = 0.0
"error-dot" = +2.5: "negative" = 0.0, "zero" = 0.50, "positive" = 0.50

1. If (e < 0) AND (er < 0) then Cool 0.00 & 0.00 = 0.0
2. If (e = 0) AND (er < 0) then Heat 1.00 & 0.00 = 0.0
3. If (e > 0) AND (er < 0) then Heat 0.00 & 0.00 = 0.0
4. If (e < 0) AND (er = 0) then Cool 0.00 & 0.50 = 0.0 >>>>
5. If (e = 0) AND (er = 0) then No_Chng 1.00 & 0.50 = 0.50
6. If (e > 0) AND (er = 0) then Heat 0.00 & 0.50 = 0.0

7. If (e < 0) AND (er > 0) then Cool 0.00 & 0.50 = 0.0 >>>>
8. If (e = 0) AND (er > 0) then Cool 1.00 & 0.50 = 0.50
9. If (e > 0) AND (er > 0) then Heat 0.00 & 0.50 = 0.0

"negative" = (R1^2 + R4^2 + R7^2 + R8^2) (Cooling) = (0.0^2 + 0.0^2 + 0.0^2 + 0.50^2)^.5 = 0.50
"zero" = (R5^2)^.5 = (0.5^2)^.5 (No Change) = 0.5
"positive" = (R2^2 + R3^2 + R6^2 + R9^2) (Heating) = (0.00^2 + 0.00^2 + 0.00^2 + 0.00^2)^.5 = 0.000

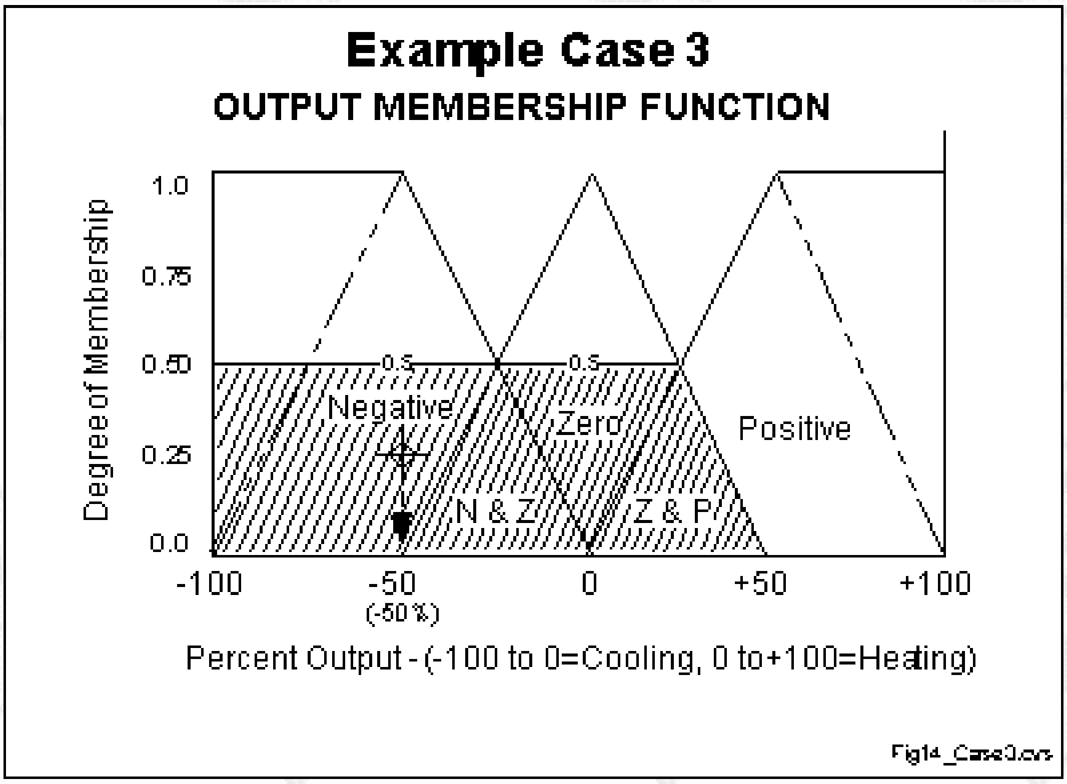


Figure 14 - Output function

(neg_center * neg_strength + zero_center * zero_strength + pos_center * pos_strength) = OUTPUT
(neg_strength + zero_strength + pos_strength)

(-100 * 0.50 + 0 * 0.50 + 100 * 0.000) = -50% (cooling)
(0.50 + 0.50 + 0.000)
EXAMPLE CASE 4 - ERROR = +0.5F

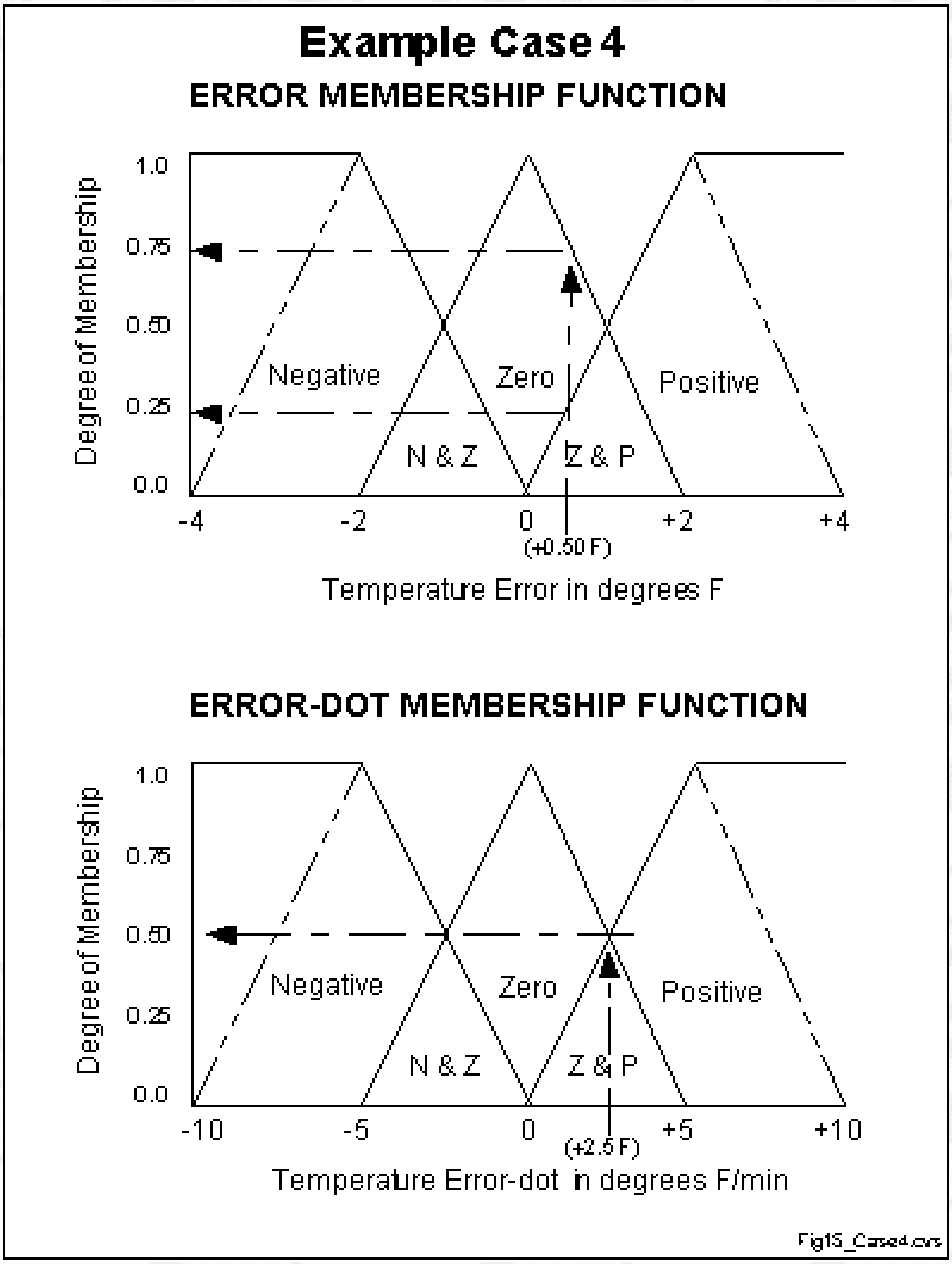


Figure 15 - EXAMPLE CASE 4

INPUT DEGREE OF MEMBERSHIP

"error" = +0.50: "negative" = 0.0, "zero" = 0.75, "positive" = 0.25
"error-dot" = +2.5: "negative" = 0.0, "zero" = 0.50, "positive" = 0.50

1. If (e < 0) AND (er < 0) then Cool 0.00 & 0.00 = 0.0
2. If (e = 0) AND (er < 0) then Heat 0.75 & 0.00 = 0.0
3. If (e > 0) AND (er < 0) then Heat 0.25 & 0.00 = 0.0
4. If (e < 0) AND (er = 0) then Cool 0.00 & 0.50 = 0.0 >>>>
5. If (e = 0) AND (er = 0) then No_Chng 0.75 & 0.50 = 0.50 >>>>
6. If (e > 0) AND (er = 0) then Heat 0.25 & 0.50 = 0.25

7. If (e < 0) AND (er > 0) then Cool 0.00 & 0.50 = 0.0 >>>>
8. If (e = 0) AND (er > 0) then Cool 0.75 & 0.50 = 0.50 >>>>
9. If (e > 0) AND (er > 0) then Heat 0.25 & 0.50 = 0.25

"negative" = (R1^2 + R4^2 + R7^2 + R8^2) (Cooling) = (0.0^2 + 0.0^2 + 0.0^2 + 0.50^2)^.5 = 0.50
"zero" = (R5^2)^.5 = (0.5^2)^.5 (No Change) = 0.50
"positive" = (R2^2 + R3^2 + R6^2 + R9^2) (Heating) = (0.00^2 + 0.00^2 + 0.25^2 + 0.50^2)^.5 = 0.354

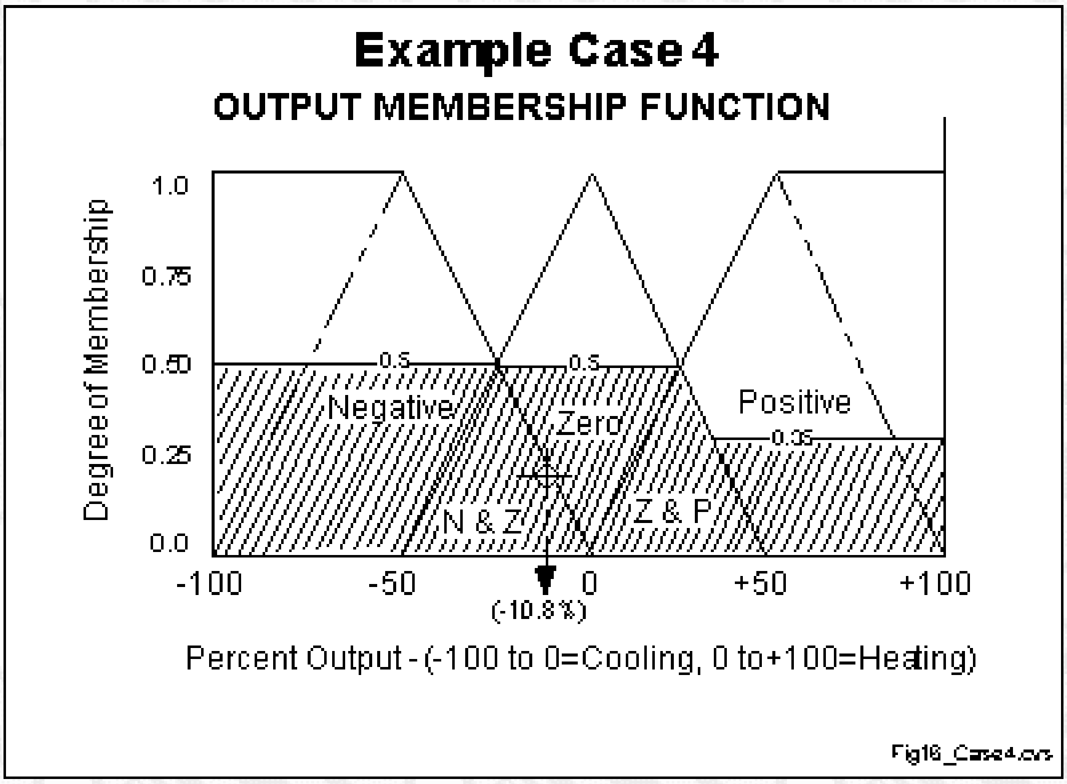


Figure 16 - Output function

<u>(neg_center * neg_strength + zero_center * zero_strength + pos_center * pos_strength)</u> = OUTPUT
(neg_strength + zero_strength + pos_strength)

<u>(-100 * 0.50 + 0 * 0.5 + 100 * 0.354)</u> = -10.8% (cooling)
(0.50 + 0.50 + 0.354)
EXAMPLE CASE 5 - ERROR = +1.0F

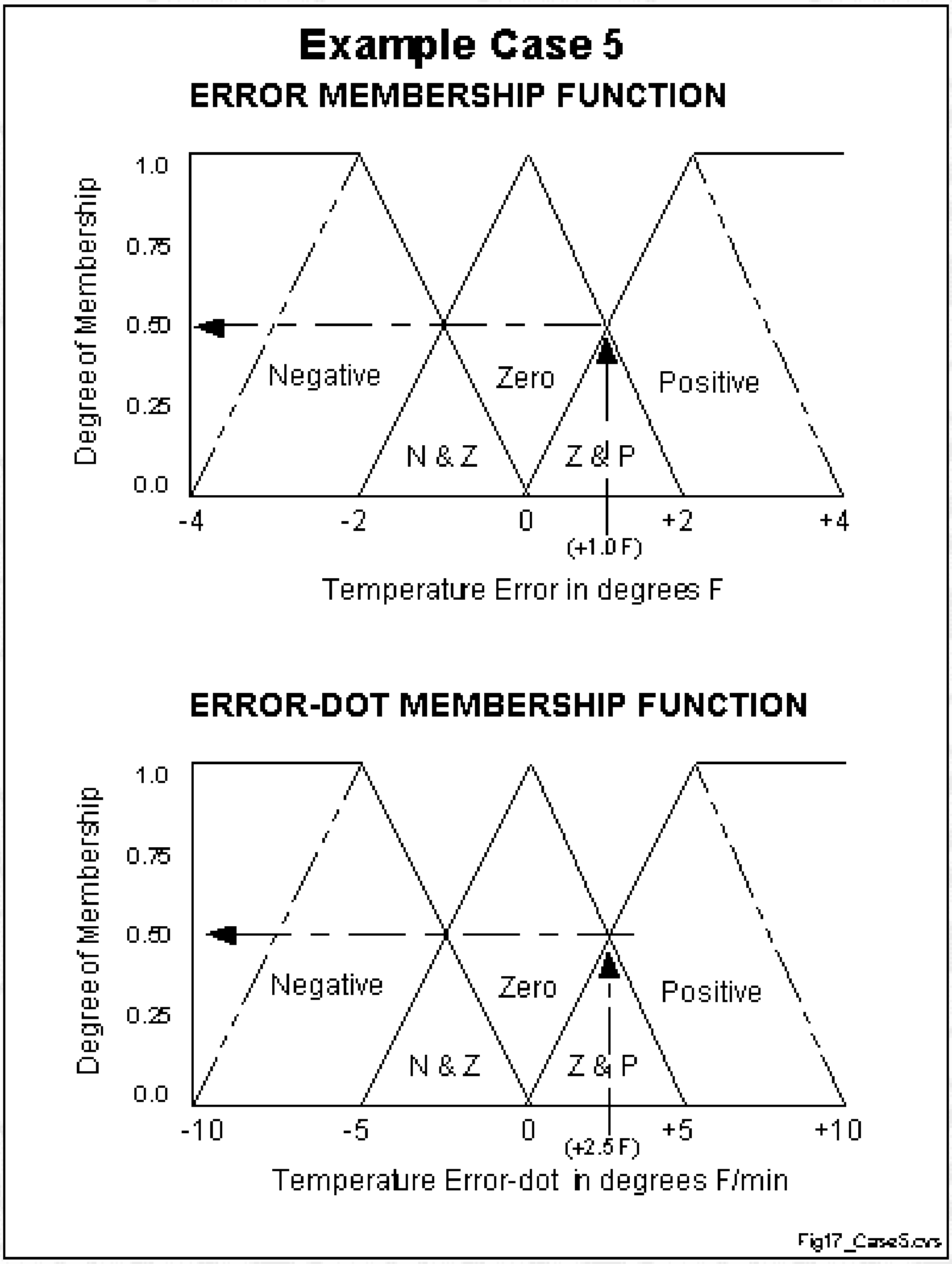


Figure 17 - EXAMPLE CASE 5

INPUT DEGREE OF MEMBERSHIP

"error" = +1.0: "negative" = 0.0, "zero" = 0.5, "positive" = 0.5
"error-dot" = +2.5: "negative" = 0.0, "zero" = 0.5, "positive" = 0.5

1. If (e < 0) AND (er < 0) then Cool 0.00 & 0.00 = 0.0
2. If (e = 0) AND (er < 0) then Heat 0.50 & 0.00 = 0.0
3. If (e > 0) AND (er < 0) then Heat 0.50 & 0.00 = 0.0
4. If (e < 0) AND (er = 0) then Cool 0.00 & 0.50 = 0.0 >>>>
5. If (e = 0) AND (er = 0) then No_Chng 0.50 & 0.50 = 0.5 >>>>
6. If (e > 0) AND (er = 0) then Heat 0.50 & 0.50 = 0.5

7. If (e < 0) AND (er > 0) then Cool 0.00 & 0.50 = 0.0 >>>>
8. If (e = 0) AND (er > 0) then Cool 0.50 & 0.50 = 0.5 >>>>
9. If (e > 0) AND (er > 0) then Heat 0.50 & 0.50 = 0.5

"negative" = (R1^2 + R4^2 + R7^2 + R8^2) (Cooling) = (0.00^2 + 0.00^2 + 0.00^2 + 0.50^2)^.5 = 0.500
"zero" = (R5^2)^.5 = (0.50^2)^.5 (No Change) = 0.500
"positive" = (R2^2 + R3^2 + R6^2 + R9^2) (Heating) = (0.00^2 + 0.00^2 + 0.50^2 + 0.50^2)^.5 = 0.707

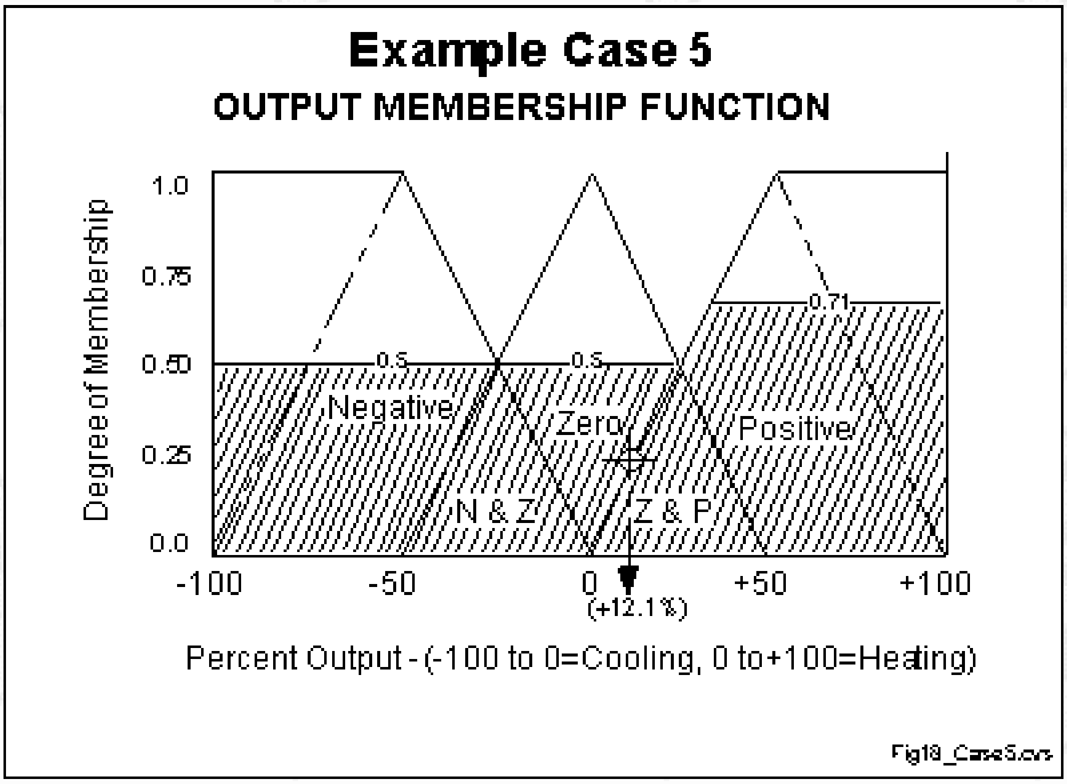


Figure 18 - Output function

(neg_center * neg_strength + zero_center * zero_strength + pos_center * pos_strength) = OUTPUT
(neg_strength + zero_strength + pos_strength)

(-100 * 0.500 + 0 * 0.500 + 100 * 0.707) = +12.1% (heating)
(0.500 + 0.500 + 0.707)

- Back to Part 6

Back to Index

ABOUT THE AUTHOR. Steven D. Kaehler is a 1993 suma cum laude graduate of Cogswell College North (now Henry Cogswell College) in Kirkland, Washington and is currently employed as an electrical engineer with the Boeing Company in Seattle WA. He has over 13 years of combined technical and engineering experience in static fatigue, environmental, and fuel systems test, measurement, and control. He has worked for the Environmental Test Laboratories organization of Boeing's Defense and Space Group (BD&SG now ISDS) in Kent, WA and currently works in the Propulsion Instrumentation division of the Airplane Systems Labs (ASL) organization of Boeing's Commercial Airplane Group (BCAG) at Boeing Field. He specializes in the design, construction, and operation of custom test instrumentation and control systems and dabbles in robotics for fun.

The Boeing Company
P.O. Box 3707 M/S 17-PA
Seattle, WA 98124
Voice: (206) 655-3921
Email: steven.d.kaehler@boeing.com

To Index

To Part 1

To Part 2

To Part 3

To Part 4

To Part 5

To Part 6

To Sample Cases