Mamdani FIS for Decoherence Rate

Contents

1 Introduction 3

2 Literature Review 3

2.1 The Concept of Fuzzy logic 3

2.2 Implementation of set Theory 3

2.2.1 Crisp Sets 3

2.2.2 Fuzzy Sets 3

2.3 Rules Base 4

2.4 Creating Realistic Boundaries 5

2.5 Mamdani FIS process 5

2.5.1 Step 1 – Fuzzification 5

2.5.2 Step 2 – Computation of Rule Firing Strength 6

2.5.3 Step 3 – Implication into Consequents 6

2.5.4 Step 4 – Combination of Consequents 6

2.5.5 Step 5 – Defuzzification 6

3 Example System 7

3.1 Defuzzification methods 8

3.2 My FIS 8

4 System Design 8

4.1 Types of Membership functions 8

4.2 Input Variables 9

4.3 Output Variables 9

4.4 Rules Base 9

5 Implementation and Development 9

5.1 Tools 9

5.2 Initializing the FIS and Input/Output Variables 10

5.3 Defining Membership Functions 10

5.3.1 Input: Temperature 10
5.3.2 Input: Magnetic Field Strength (MFS) 10
5.3.3 Input: Noise Level 11
5.3.4 Output: Decoherence Rate 11
5.4 Rules Base 11
5.5 Visualizing Variables and Fuzzy Rules List 11
6 Testing and Evaluation 11
6.1 Theoretical Data fuzzified. 11
6.2 Theoretical Data Output Results 12
6.3 Comparison to Real-world Findings 13
6.4 Evaluation 14
6.4.1 Accuracy 14
6.4.2 Reliability 14
6.4.3 Limitations 14
7 References 15
8 Appendix A: Python Code for Rule Generation 16
9 Appendix B: Theoretical Dataset 16

1 Introduction

This report outlines the development of a Fuzzy Inference System (FIS) in MATLAB to estimate the decoherence rate in quantum systems. Decoherence—caused by environmental factors—represents the loss of quantum coherence in qubits, leading to potential data corruption or loss. The FIS models this effect using fuzzy logic, allowing nuanced classification of imprecise inputs. The system takes three real-world input variables—Temperature, Magnetic Field Strength, and Noise Level—and produces a single output: Decoherence Rate. Initial implementation includes theoretical and real-world testing. The system may be expanded to enhance realism based on evaluation outcomes.

2 Literature Review

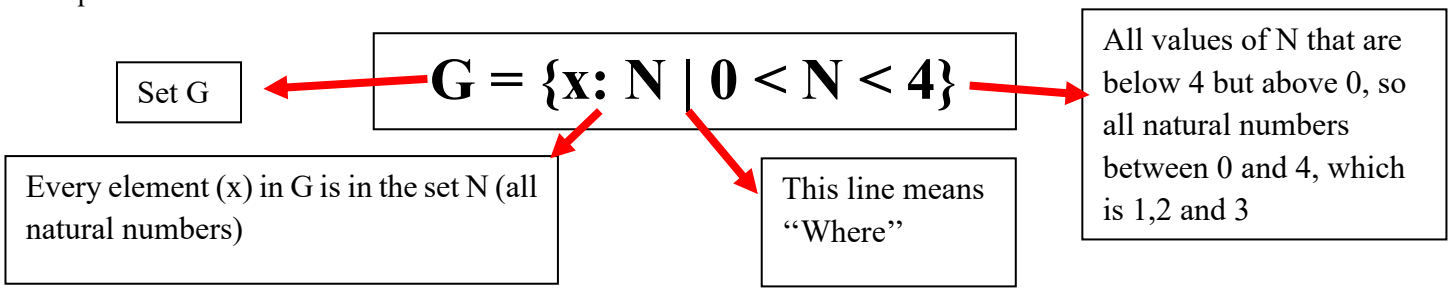
2.1 The Concept of Fuzzy logic

In real life, classification of objects and forces are commonly abstract and lack objective boundaries. For example, 20 degrees Celsius in a room could be considered 'hot' to some but 'warm' or even 'cold' to others. In a FIS system, we take that into account and fuzzify the outputs. This means if the question 'is this room that is 20 degrees Celsius hot?' is queried, instead of 'yes' or 'no' the system replies with a degree of membership that 20 degrees belongs in the set called 'hot'. (This will be explained in greater detail later).

2.2 Implementation of set Theory

2.2.1 Crisp Sets

A crip set is a well-defined collection of distinct objects. For example, set G may contain the elements 1,2,3. The set is expressed:



This expression means 'All elements (x) of Set G are in set N (all natural numbers) where N is above 0 and below 4.

Crisp Sets have objective and defined boundaries that cannot be interpreted, ergo the name 'Crisp' to declare the clearness and precision of elements belonging to the set. However, a FIS uses 'Fuzzy Sets'

2.2.2 Fuzzy Sets

In Fuzzy sets, elements no longer have a binary value of whether they belong in the set, but instead they have a degree of membership. Meaning, an element now has a *strength* or *amplitude* of belonging in the set. The degree of membership is commonly normalised and will be for the entirety of this project (meaning the highest degree of membership an element can have been 1). Let's now assume that Set G that was mentioned previously is a fuzzy set. This means if we ask, 'does 2 belong in set G?' we would express as the following:

$$\mu G(2) = 0.5$$

The symbol μ denotes a membership function. It outputs the degree to which a value belongs to a fuzzy set. In the case of the equation on the left, 2 has a 0.5 membership value of belong to fuzzy set G. This does not represent a probability — it's not a 50% chance of being in the set. Rather, it expresses a degree of membership: a 0.5 out of 1 strength of association with the fuzzy concept represented by G.

It is common practice to visualize fuzzy sets on a graph. (the specifics of how this is programmed will be discussed further in this report). The blue line represents the membership function of fuzzy set G, which maps each input value (X-axis) to a degree of membership (Y-axis). This function visually defines the fuzzy set. The function itself is called μG

- The X-axis represents the values that belong to the function (the element)
- The Y-axis represent the degree of membership that value is a member of the function

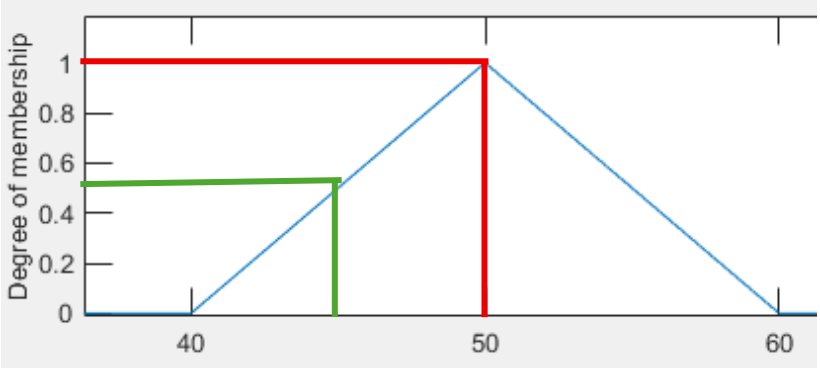


Figure 1.1

For example:

- For value 50, the degree of membership that is a member of the function A is 1 (see the red lines).
- For the value 45, the degree of membership it is a member of function A is 0.5 (see the green lines).

In a mathematical formula, this can be expressed as:

μG(50) = 1 μG(45) = 0.5 μG(38) = 0

2.3 Rules Base

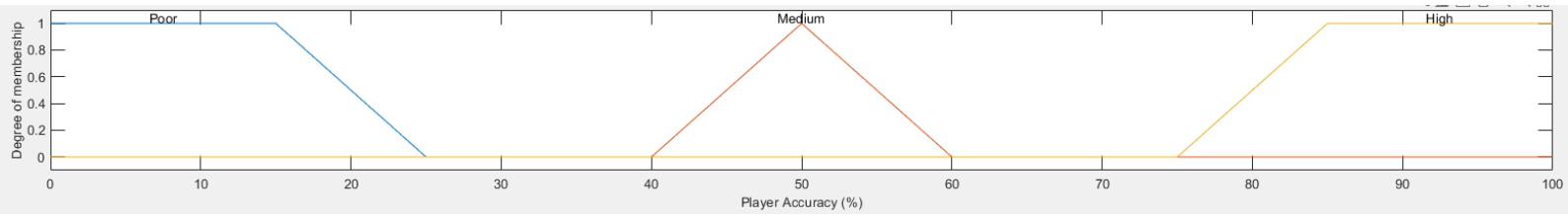


Figure 3.2

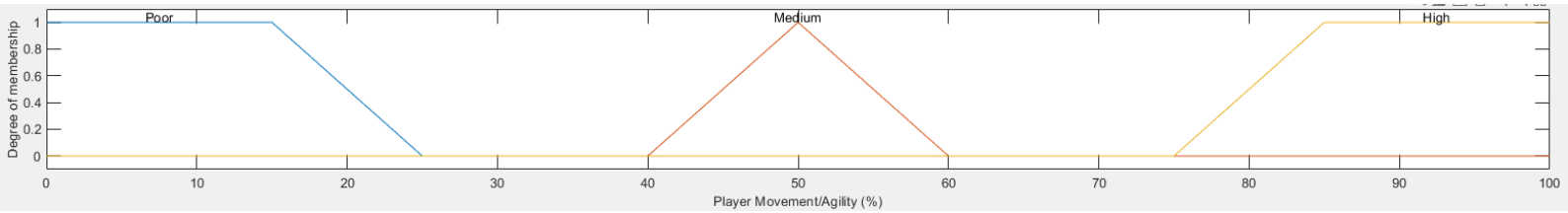


Figure 4.3

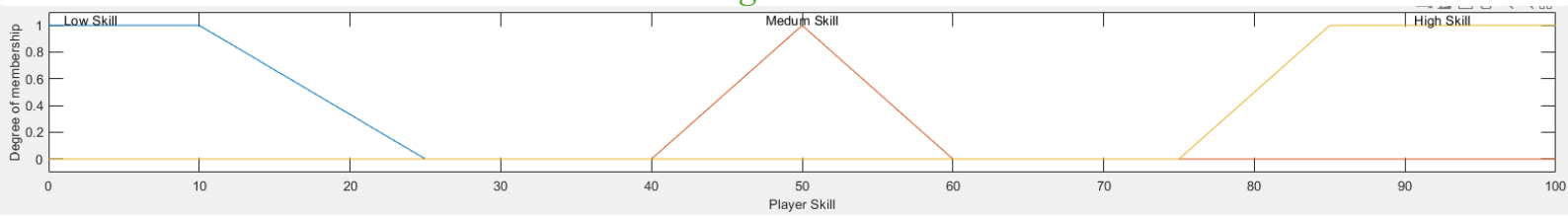


Figure 2.4

A 'rule' is what utilities inputs to produce an output variable, and a rules base is just the collection of these rules. This is shown in the examples above. Generally, increasing the number of rules in the rule base typically leads to higher accuracy in the results.

Here we have input variables 'Player Accuracy %' (figure 1.2), 'Player Movement/Agility (%)' (figure 1.3) and an output variable 'Player Skill' (figure 1.4). To generate an output from an input is using a rule. Here is an example of a rules base:

1. If (Player Accuracy (%) is Poor) AND(min) (Player Movement/Agility (%) is Poor) then (Player Skill is Low Skill)
2. If (Player Accuracy (%) is Medium) AND(min) (Player Movement/Agility (%) is Medium) then (Player Skill is Low Skill)
3. If (Player Accuracy (%) is Poor) AND(min) (Player Movement/Agility (%) is High) then (Player Skill is Medum Skill)
4. If (Player Accuracy (%) is High) AND(min) (Player Movement/Agility (%) is Medium) then (Player Skill is Medium Skill)

The role of AND(min) is detailed in 2.5.2. Focusing on rule 3, if the player accuracy is found in Poor function, and the player movement value is found in the High function, the output will be found in the medium function. This is the foundation of how a rule base operates. Now we have seen how a FIS can allow inputs and produce outputs.

2.4 Creating Realistic Boundaries

when mapping multiple membership functions, we can create:

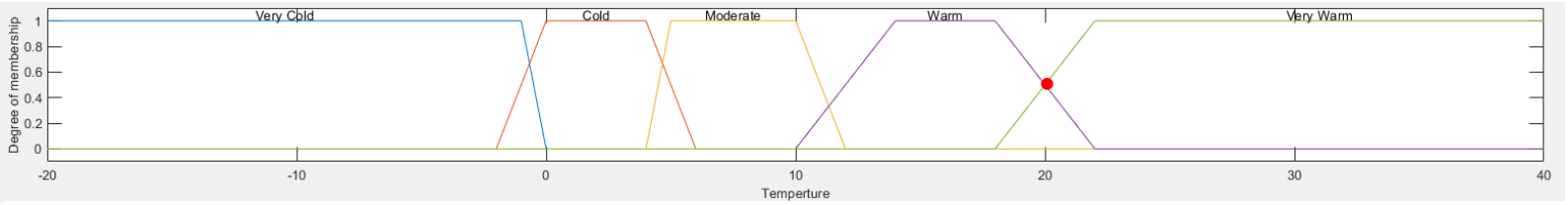


Figure 5.5

Here we see 5 membership functions. Very Cold, Cold, Moderate, Warm, Very Warm. You can see they overlap, which represents realistic fuzziness of the sets they represent. For example, on the graph above, 20 degrees (which is presented by a red dot) has a degree of membership in both 'Warm' and 'Very Warm'. Initially, this sounds like breaking the law of contradiction, which is defined as:

μWarm(20) = 0.5 μVeryWarm(20) = 0.5

- **a proposition and its negation cannot both be true at the same time and in the same sense**

A temperature cannot be both warm and very warm because if the temperature is 'very warm' then it's not warm, and vice versa. This is a key component to fuzzy sets; they break the law of contradiction. This leads to a greater realistic approach to mapping out abstract classifications, such as warm and very warm. Since 20 degree is not objectively 'warm' or 'very warm', it can belong to both classifications. A concept that is extremely difficult to model in traditional set theory.

The collection of these five membership functions is called a 'fuzzy partition'. When a temperature value is provided to the FIS, it will have degrees of membership in one or more of these overlapping functions. This fuzzy representation allows for more realistic modelling of ambiguous concepts like temperature.

2.5 Mamdani FIS process

There are different types of these systems, one of which is a Mamdani type. This defines the process of the FIS, more specifically, it determines how rules are formulated, inference is performed, and outputs are generated. The Mamdani process has 5 distinct inference steps.

2.5.1 Step 1 – Fuzzification

All input values are fuzzified, and their membership grade is calculated using membership functions, for example seen in section 2.2.2, the crisp values 50,45 and 38 are fuzzified to 1,0.5 and 0.

2.5.2 Step 2 – Computation of Rule Firing Strength

The rules in the rule base are applied by assessing how well each input combination satisfies the corresponding rule conditions. For example, we identify which rules listed in Section 2.4 are activated using the crisp data (Player Accuracy, Player Movement) = [(7, 24),(29,80),(80,55)]. Focusing on the first data pair (7,24)

- In Figure 1.2 μPoor(7) = 1 , μMedium(7) = 0 , μHigh(7) = 0
- In Figure 1.3 μPoor(24) = 0.1 , μMedium(24) = 0 , μHigh(24) = 0

Focusing on rule 1 : If (Player Accuracy (%) is Poor) AND(min) (Player Movement/Agility (%) is Poor) then (Player Skill is Low Skill)

The AND operator requires both input fuzzy values to be greater than 0. The min operator is then used to calculate the **firing strength** of the rule, by selecting the **lower** of the two membership values. This can come in the form of OR(min) too, which does allow input fuzzy values to be 0 and still fire off the rule. The min operator still acts the same. This firing strength determines how strongly the rule contributes to the final output. Therefore Rule 1:[7,24] = 0.1. (7, 24) does not satisfy rule 2,3 or 4. (29,80) does not satisfy any rule. (80,55) only satisfied rule 4 with firing strength 0.5.

2.5.3 Step 3 – Implication into Consequents

In this step, the output fuzzy set (the consequent) for each activated rule is scaled according to the firing strength calculated in Step 2. While rule 2 and rule 3 provides no effect: Rule 1 provides an output low skill fuzzy set, limited to a maximum membership degree of 0.1 (min operator again). Rule 4 provides an output medium skill fuzzy set, limited to a maximum membership degree of 0.5 (min operator again). The Mamdani process typically uses the min operator in step 3.

2.5.4 Step 4 – Combination of Consequents

The rule outputs are aggregated by combining the fuzzy output sets from all the rules into a single fuzzy set. This is done using the OR operator. (This is shown visually Figure 1.8 in Section 3)

2.5.5 Step 5 – Defuzzification

Return a Crisp singleton value. There are several different methods for this process, as listed in section 3.1.

3 Example System

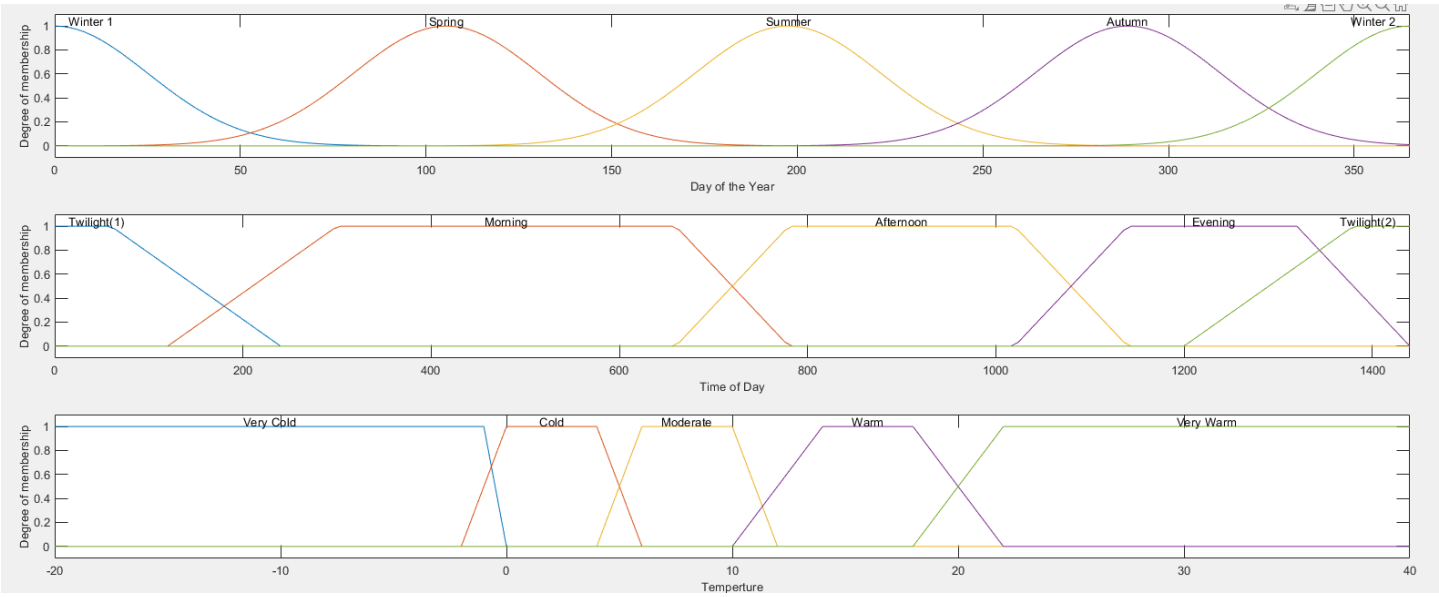


Figure 7.6

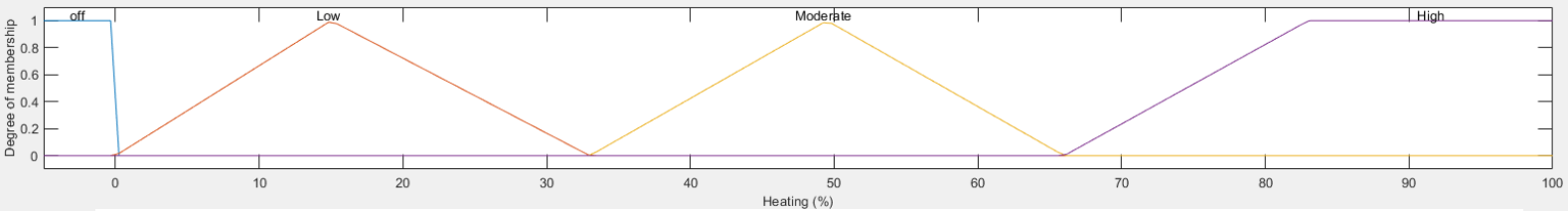


Figure 6.7

To implement the concepts introduced in Section 2, we construct an example system in MATLAB. In this environment, fuzzy rules are represented in vector form. For instance, **Rule1 = [1 1 1 1 1]**.

- The 1st number = Membership function of the first input variable (function 'Very Cold' in input 'Temperature')
- The 2nd number = Membership function of the second input variable (function 'Low' in input 'Magnetic Field Strength' - MFS)
- The 3th number = Membership function of the output variable (function 'Low' in output 'Decoherence Rate')
- The 4th number = rule weight (between 0 – 1, default = 1)
- The 5th number = the operator type (1 = OR, 2 = AND)

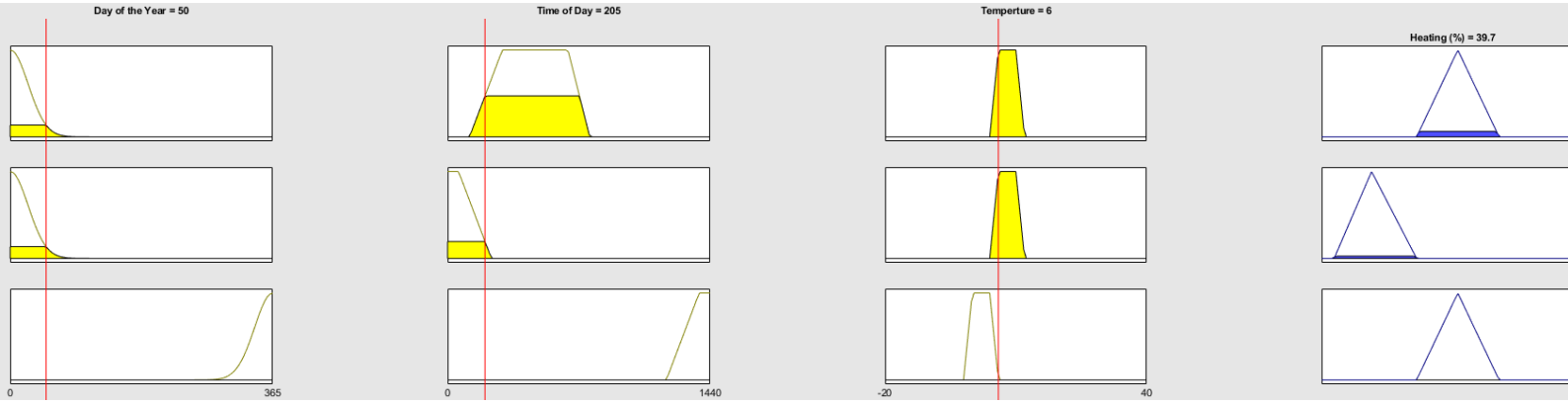


Figure 8.8

Here we have a system with Input variables 'Day of the Year', 'Time of Day', and 'Temperature'(figure 1.6) . And produces the output 'Heating%' (figure 1.7). Let us assume the rules in the rule base are Rule1 = [1 2 3 3 1 1];, Rule2 = [1 1 3 2 1 1]; , Rule3 = [5 5 2 3 1 1]; If we input the values Day of the year = 50, Time of Day = 205, Temperature = 6, we can calculate the degree of membership for all membership functions and the use the appropriate ones the rules refer

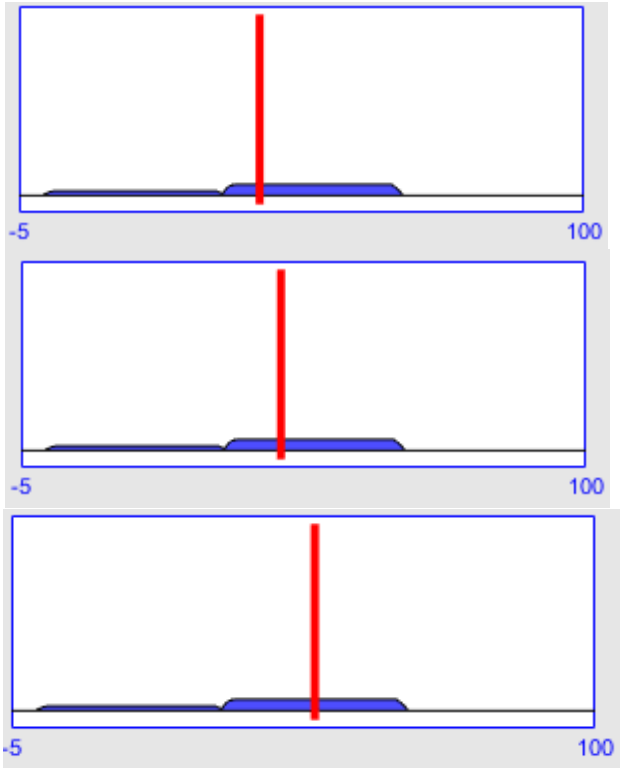
to. This is fuzzifying the crisp values to fuzzy values. For rule 1, we have (0.1353, 0.4722, 1) and since the OR(min) operator is used, the firing strength is 0.1353. After multiplied by the rule weight (1), the strength remains at 0.1353. Copy for both rule 2 and 3, and we get values 0.1353 and 0. Next is to map these degrees of membership onto the temperature graph. In figure 1.8, the rows represent the rules, the input variables in yellow and output variable in blue. The bottom right image shows the overall result of the inputs and the combination of the rules. From this, we can retrieve an arrange of values, depending on the method used.

3.1 Defuzzification methods

- Centroid = find the centre of mass, by considering the amount of area and the distribution of the mass / area. This provides the value 39.7

- Bisector = Does not consider where the mass is concentrated, only the shape of the area. It finds the middle value that splits the total area in half. This provides the value 43.3

- Mom = Identifies the largest degree of membership, then locates all x-values in this membership function that equals the maximum degree of membership. It then calculates the mean of these x-values and returns this output. This provides the value 49.6



For my FIS, the Centroid method would be best suitable due to avoiding abrupt changes and show gradual change. It is sensitive to rule overlap, reflects rule weights and is used commonly for scientific modelling and estimation.

3.2 My FIS

There are several reasons as to why I chose my specific project. A strong one is that it also links to my other project 'Quantum Simulation of Physical Simulations'. This FIS I am building can provide the settings of the environment the simulation takes place by producing a 'decoherence rate'. This is the rate in which superconducting qubits (the ones that are being simulated in the separate project) lose coherence, which can lead to unwanted changes or loss of data entirely. This rate can be altered by several factors, and in many instances these factors cannot be classified into crisp sets. Additionally, decoherence rate itself can be uncertain, and so a degree of membership may be more suitable. Along with its ability to produce many different outputs for different inputs, it is a suitable project.

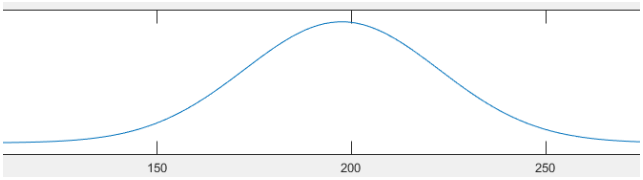
My project will inherit fuzzy logic and use it to fuzzify inputs, calculate fuzzy results using a rules base, and then defuzzify the results.

4 System Design

4.1 Types of Membership functions

There are several different types of Membership functions. The type determines how the degree of membership is distributed across the fuzzy set. Any types of membership functions used in this report will be demonstrated. The types used in this project are:

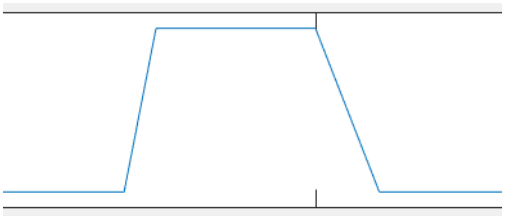
Gaussian membership Function (Gaussmf)



A Gaussian membership function allows for a gradual symmetrical decay in degree of membership. It is formed by the equation (where C = the centre and σ = standard

$$\mu(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$$

Trapezoidal Membership Function (Trapmf)



This allows a large range of values to have a degree of membership of 1, with a linear decline on both sides

4.2 Input Variables

There will be three initial inputs variables (and respected units):

- Temperature (millikelvin), - with four membership functions named Very Cold, Cold, Warm and Hot.
- Magnetic Field Strength (in millitesla) – with three membership functions named Low, Medium and High.
- Noise Level (unitless) – with three membership functions named Low, Medium and High.

All membership functions in Temperature and MFS will be Gaussian, while the ones for Nosie level will be Trapezoidal. The number of units are extremely small due to the realistic sensitivity of quantum systems, and how easily they can become decoherent by an external force. The range will be determined during implementation.

4.3 Output Variables

The output variable is 'Decoherence rate', which will be in respect of time (per nanosecond). with a range of 0 – $1x10^{-3}$. It will have three membership functions named Low, Medium and High.

4.4 Rules Base

To generate the rule base, all possible combinations of input membership functions are considered, resulting in 36 rules (4 × 3 × 3). Each rule applies the OR operator, reflecting the independence of the three inputs—temperature, MFS, and noise level—each of which can independently cause decoherence. This mirrors real-world quantum systems, where a single unfavourable condition can significantly affect stability.

In this initial phase, all rules are assigned equal weights (1) to maintain simplicity and model stability. Although temperature typically has the greatest impact, followed by MFS and then noise (which varies by type), using equal weights avoids early over-sensitivity. As Nielsen and Chuang note, "thermal interactions with the environment are among the primary causes of decoherence in quantum systems" (Nielsen & Chuang, 2010). Future iterations may explore weighted rules to improve accuracy.

5 Implementation and Development

This section of the report will detail the implementation and development process of the project, outlining the key design decisions, technologies used, and steps taken to build and refine the system. All text in red is code directly from the program

5.1 Tools

The FIS will be developed in MATLAB due to its robust built-in tools for designing, visualizing and analysing fuzzy logic systems. These include:

1.) Subplot = used to visualize the membership functions of both input and output variables, illustrating the overlaps between functions and the relationship between crisp input values (x-axis) and their degrees of membership (y-axis).
2.) ruleview = a graphical interface that enables detailed inspection of how each rule in the rule base is triggered by the inputs and how they contribute to the final output.

Additionally, Python will be used to assist in generating the fuzzy rules, leveraging its concise syntax and efficient use of for loops to automate rule creation.

5.2 Initializing the FIS and Input/Output Variables

The first step in building the fuzzy inference system is to define its structure. A Mamdani-type system is used for this project, which is initialized in MATLAB using the mamfis() function. The Name parameter assigns a label to the FIS, as shown: A = mamfis('Name', 'FIS Decoherence Rate'); The variable A stores the FIS object and is used throughout the development process when assigning membership functions, input/output variables, and fuzzy rules.

Next is to declare the input/output variables to the FIS we initialized. We use addInput() and addOutput() functions to create them and then assign membership functions to them. Inside these functions we pass the parameters of the FIS, range and name). For example, the Temperature input it is declared as: A = addInput(A, [5 150], 'Name', 'Temperature (millikelvin)');

The reason there is an A = is because MATLAB functions often return a modified version of the object, and you need to reassign it to keep those changes. So now A is an updated version of the FIS that includes the input variable. [5 150] will be the values we see on the X-axis on the graph (as we see in section 3's graphs).

Inputs named 'Magnetic Field Strength (millitesla)' is created with parameters A, [0 10], and 'Noise Level (Unitless)' with parameters A, [0 1]. And the output variable is named 'Decoherence Rate (per nanosecond)' with parameters A and [0 0.001].

5.3 Defining Membership Functions

The addMF() function is used to create the membership functions. They are defined in the following form:

addMF(FIS, Variable_Name, Membership_Type, Membership_Values, 'Name', Function_Name) , where:

1. FIS - the fuzzy inference system object to which the membership function is being added.
2. Variable_Name – Name of the input/output variable the membership function nis assigned to.
3. Membership_Type – Type of membership function used to shape the fuzzy set. Such as those listed in section 4.1
4. Membership_values – A vector defining the shape of membership functions. The values of number of values depend on the Membership_Type.
    - Gaussmf : [A, B] where A = standard deviation and B = centre
    - Trapmf: [A, B, C, D] where each set the corner of the trapezoid.
5. Function_Name – The name of this specific membership function

5.3.1 Input: Temperature

Four membership functions are assigned to the input variable ', 'Temperature (millikelvin)'. Each are assigned to the same FIS object A and are membership type Gaussmf. Each with the following parameters:

- Name 'Very Cold', with Membership values [4 5]
- Name 'Cold', with Membership values [12 50]
- Name 'Warm', with Membership values [14 100]
- Name 'Hot', with Membership values [18 150]

5.3.2 Input: Magnetic Field Strength (MFS)

Three membership functions are assigned to the input variable 'Magnetic Field Strength (millitesla)'. Each are assigned to the same FIS object A and are membership type gaussmf. Each with the following parameters:

- Name 'Low', with Membership values [1 0]
- Name 'Medium, with Membership values [1.5 4]
- Name 'High, with Membership values [1.5 10]

5.3.3 Input: Noise Level

Three membership functions are assigned to the input variable 'Noise Level (Unitless)'. Each are assigned to the same FIS object A and are membership type trapmf. Each with the following parameters:

- Name 'Low', with Membership values [0 0 0.2 0.4]
- Name 'Medium', with Membership values [0.3 0.5 0.5 0.7]
- Name 'High, with Membership values [0.6 0.8 1 1]

5.3.4 Output: Decoherence Rate

Three membership functions are assigned to the output variable 'Decoherence Rate (per nanosecond)'. Each are assigned to the same FIS object A and are membership type gaussmf. Each with the following parameters:

- Name 'Low', with Membership values [0.0001 0]
- Name 'Medium, with Membership values [0.0001 0.0005]
- Name 'High, with Membership values [0.0002 0.001]

5.4 Rules Base

As stated in section 4.4, the rules base will consist of 36 rules, the maximum number of combinations we can create with the number of input membership functions. A simple IDLE script titled 'FIS_CREATE_RULES' in python was created for this. Using three nest for-loops

For A in range of 1 to 4: (to iterate through the membership functions of Input 1 – Temperature):

For B in range of 1 to 3: (to iterate through the membership functions of Input 2 – Magnetic Field Strength):

For C in range of 1 to 3: (to iterate through the membership functions of Input 3 – Noise Level):

Each rule is structured as [A B C D 1 1], where D = (A + B + C) / 3, assigning an averaged output membership index to reflect the combined influence of inputs. The rules are compiled into a matrix:

ruleList = [Rule1; … Rule36]; , which is passed into the addRule() function to populate the FIS with the rules. This is done by : A = addRule(A, ruleList);.

5.5 Visualizing Variables and Fuzzy Rules List

To visualize and interact with all the rules in the fuzzy rules set we use the ruleview() function, with FIS object (A) as the parameter. This provides us the window we see in figure 1.8 in section 3 for all 36 rules. For graphs like figures 1.2 – 1.7, we use the subplot() and plotmf() function. Since we have 4 variables (inputs and outputs combined) we will set the parameters as subplot(4,1,x). This divides the window into 4 rows and 1 column, placing each plot in position x (from top to bottom). x will vary from 1 to 4. On the same line as the subplot, the plotmf() function is written. This draws the membership function for the specified variable in FIS object A.

For example, to plot input variable 'Temperature': subplot(4,1,1),plotmf(A, 'input', 1) – this command displays the membership functions of input variable 1 (i.e., the first input added to FIS object A) in the top subplot of a 4-row figure layout. Now the program is created, we can test and evaluate the results produced.

6 Testing and Evaluation

There are several components to test, first is to see if crisp input values being fuzzified correctly. Next is testing theoretical data produces correct fuzzified and defuzzied output results. The final part is to test real-word data produces correct crisp output results.

6.1 Theoretical Data fuzzified.

In the program under the heading '--------- TESTING THEORETICAL DATA ' , tests the FIS using synthetic theoretical data to verify correct fuzzification. Ensures each membership function assigns correct degrees of membership to the crisp input values. The test data consists of a 3x10 matrix variable titled 'Theoretical_Data' , each row representing a set of crisp inputs in the order : temperature, MFS, noise level. The logic of the program operates as follows:

For each row in Theoretical_Data (3x5 matrix):

Assign current row to variable x

For each input variable in the fuzzy system:

Print the name of the input variable and the current crisp input value

For each membership function associated with this input:

Get the function name, type (**gaussmf** or **trapmf**) and parameters

Use **feval()** function to evaluate the membership degree for the input value

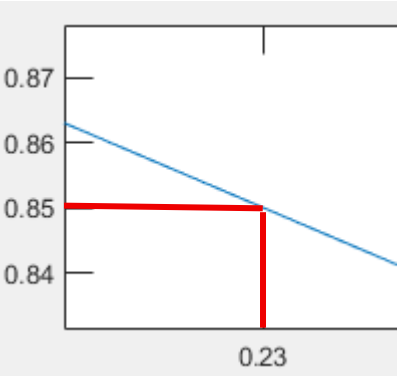
Print the result "μ_functionName(value) = degree"

The purpose of this code confirms correct fuzzification of crisp input values, as well as all membership functions are accessed and evaluated. Below are the results from Test case 4 (135.27, 9.48, 0.23):

===== Test Case 4 =====
Evaluating membership functions for input variable: Temperture (milliKelvin) (x = 135.27)
μ_Very Cold(135.27) = 0.0000
μ_Cold(135.27) = 0.0000
μ_Warm(135.27) = 0.0419
μ_Hot(135.27) = 0.7155
Evaluating membership functions for input variable: Magnetic Field Strength (milliTesla) (x = 9.48)
μ_Low(9.48) = 0.0000
μ_Medium(9.48) = 0.0013
μ_High(9.48) = 0.9417
Evaluating membership functions for input variable: Noise Level (Unitless) (x = 0.23)
μ_Low(0.23) = 0.8500
μ_Medium(0.23) = 0.0000
μ_High(0.23) = 0.0000

With the Noise level values, we can easily measure it with the ruleview() window due to the straight lines the membership functions create. For the gaussian function for Temperture and MFS we can use the equation in section 4.1.

$$\mu(x) = e^{-\frac{(x-c)^2}{2\sigma^2}}$$



- For temperature μ_Warm MF: x = 135.27, c = 100 and σ = 14
- For temperature μ_Hot MF: x = 135.27, c = 150 and σ = 18
- For MFS μ_ Medium: x = 9.48, c = 4, σ = 1.5
- For MFS μ_ High: x = 9.48, c = 10, σ = 1.5

The equation gives us the exact same answers we see produced by the program, providing evidence the program is operating correctly.

6.2 Theoretical Data Output Results

generate crisp output values. If these outputs align with our expected results, it will confirm that the system is correctly fuzzifying the input variables and processing them accurately through the fuzzy inference system. This is achieved using MATLAB's evalfis() function, which takes the FIS object A and the matrix of input values Theoretical_Data as parameters. The resulting crisp outputs are stored in the variable Theory_Crisp_Outputs as follows:

Theory_Crisp_Outputs = evalfis(A, Theoretical_Data);

Using this variable, we apply a for loop as follows:

For each value in Theory_Crisp_Outputs:

Get its index (text case number) and matching input row from Theoretical_Data

Print test case number, inputs and output in formatted line

From the code under the title '% Theoretical Data Output Results' the results produced are as follows:
Crisp Input 1: [25.00, 0.50, 0.50] --> Crisp Output = 0.000500
Crisp Input 2: [50.00, 0.70, 0.10] --> Crisp Output = 0.000197
Crisp Input 3: [18.00, 4.00, 0.90] --> Crisp Output = 0.000501
Crisp Input 4: [135.27, 9.48, 0.23] --> Crisp Output = 0.000810
Crisp Input 5: [63.91, 2.15, 0.63] --> Crisp Output = 0.000521
Crisp Input 6: [89.34, 7.73, 0.04] --> Crisp Output = 0.000502
Crisp Input 7: [14.66, 0.18, 0.87] --> Crisp Output = 0.000500
Crisp Input 8: [141.89, 3.02, 0.31] --> Crisp Output = 0.000525
Crisp Input 9: [46.22, 5.57, 0.96] --> Crisp Output = 0.000505
Crisp Input 10: [101.78, 1.13, 0.44] --> Crisp Output = 0.000512

Focusing on Crisp Input 1: (and following the same process as in section 3)

- Temperture = 25 is fuzzified to membership function (MF) 2 with a degree of 0.1142.
- Magnetic Field Strength = 0.5 is fuzzified to: MF 1 = 0.8825 and MF 2 = 0.0657.
- Noise Level = 0.5 is fuzzified to MF 2 with a degree of 1.

This combination activates two rules. [2 1 2 2 1 1] and [2 2 2 2 1 1] and creates the firing strengths 0.1142 and 0.0657. Both rules contribute to the same output MF (Medium) and shape the output fuzzy set by clipping it at their respective firing strengths (as see in section 3.1).
To compute the final crisp output, the system uses the Centroid Defuzzification Formula:

x_i = crisp output values sampled from the output do

$\mu(x_i)$ = The aggregated degree of membership at that output value

$$\text{Crisp Output} = \frac{\sum_i x_i \cdot \mu(x_i)}{\sum_i \mu(x_i)}$$

Since both rules contribute to the same output MF, xi = 0.0005 (the centre value of the Medium MF in output variable 'Dechoerence rate). This is also because the formula computes the centre of gravity of the fuzzy output set.

$\mu(x_i)$ = [0.1142, 0.0657] . Now when we run the equation $\frac{(0.0005*0.1142)+(0.0005*0.0657)}{0.1142+0.0657} = 0.0005$

If we run this process for all other 9 test cases, we get the results we see produced by the system. So now we have proven the program produces correct crisp output values.

6.3 Comparison to Real-world Findings

To strengthen the real-world relevance of our program, it is important to compare its behaviour with established findings in the academic literature.

- 2.Takahashi et al. (2011) observe that "increasing the temperature increases the decoherence rate, a behaviour consistent with the increase of phonon population and thermal activation of fluctuators." This aligns closely with the dynamics we have already implemented in our Fuzzy Inference System (FIS).
- Similarly, 3.Quantum Decoherence and the Measurement Problem (n.d.) states, "The decoherence rate depends on several factors including temperature, uncertainty in position, and number of particles surrounding the system. Temperature affects the rate of blackbody radiation – each radiated photon will interact with the environment." This highlights how elevated temperatures amplify environmental interactions through blackbody radiation, thereby accelerating decoherence—an effect also captured in our model.
- Finally, 4.Montina and Arecchi (2008) note that "the well-known increase of the decoherence rate with the temperature, for a quantum system coupled to a linear thermal bath, no longer holds for a different bath dynamic." While their study explores exceptions under nonlinear bath dynamics, it nonetheless reaffirms the conventional understanding that higher temperatures generally promote decoherence in systems with linear environments.

Together, these sources validate the temperature-dependent behaviour our system models.
To validate the accuracy of the 'Noise Level' variable in our system, we reference key findings from the literature:

- 5.Jafari et al. (2025) report that "decoherence due to the nonequilibrium critical dynamics of the environment is amplified in the presence of uncorrelated and correlated Gaussian noise," confirming that greater noise levels enhance decoherence.
- Similarly, 6.Bergli, Galperin, and Altshuler (2009) state that "the revivals decay and scale exponentially with noise intensity," further reinforcing the direct relationship between increased noise and accelerated decoherence.

To ensure the accuracy of our Magnetic Field Strength (MFS) input, we refer to findings by 2.Takahashi et al. (2011), who observed that increasing the magnetic field strength tends to reduce the rate of decoherence. This is a commonly reported trend in many quantum systems, where stronger magnetic fields help stabilise the system against environmental disturbances. In contrast, our current FIS model assumes that higher magnetic field strength increases decoherence, which contradicts these findings. This inconsistency highlights an area for improvement, and future revisions of the system will adjust the relationship to better reflect the observed inverse correlation.

6.4 Evaluation

6.4.1 Accuracy

The accuracy of the Fuzzy Inference System (FIS) was assessed through theoretical datasets and real-world comparisons. For the theoretical data, the results when running the program matched exactly the expectations, confirming the system behaves consistently across a wide range of inputs. Further accuracy checks were conducted by comparing the system's output behaviour with findings from established academic literature. As stated in section 6.3, multiple peer-reviewed sources confirmed temperature and noise levels modelled in our system were successful replicas. However, in the case of magnetic field strength, our current implementation assumes a direct relationship with decoherence rate, which contrasts with empirical findings suggesting an inverse relationship.

6.4.2 Reliability

The reliability of the FIS is reinforced by its use of three independent input variables—temperature, magnetic field strength, and noise level—which ensures outputs reflect a broad range of environmental conditions. The rule base employs the OR operator, allowing any single input to activate a rule. This design enhances consistency by ensuring that meaningful outputs are still produced even when only one variable change significantly. Additionally, the use of Gaussian membership functions ensures smooth, gradual transitions in output. Their continuous nature prevents abrupt changes, supporting stable system behaviour across varying input values.

6.4.3 Limitations

While the FIS demonstrates strong accuracy and reliability, there are several limitations. Most notably, the current rule base assumes that all three inputs contribute equally to the decoherence rate and uses uniform rule weights (all equal rule weight 1). Temperature may have a greater impact than noise or magnetic field strength, which the system does not yet reflect.
Additionally, the model incorrectly assumes a direct relationship between magnetic field strength and decoherence, contrary to established findings. This affects the realism of outputs under certain conditions and is acknowledged as an area for future revision.
Finally, the system does not yet account for the complex interactions or non-linear behaviours that may exist between variables in real quantum environments. While the current setup is sufficient for demonstrating the concept, more advanced modelling would be needed for precise predictive use in real-world systems.

This report comprises 16 pages in total: Pages 1–2 cover the Title Page and Table of Contents; Pages 3–14 contain the main content; Page 15 lists the references; and Page 16 includes the appendices.

The total word count is 5,582.

7 References

1. Nielsen, M.A. and Chuang, I.L. (2010). Quantum Computation and Quantum Information. Cambridge University Press.

2. Takahashi, S., Tupitsyn, I.S., van Tol, J., Beedle, C.C., Hendrickson, D.N. and Stamp, P.C.E. (2011). Decoherence in crystals of quantum molecular magnets. Nature, 476(7358), pp.76–79. doi:https://doi.org/10.1038/nature10314.

3. Quantum Decoherence and the Measurement Problem. (n.d.). Available at: https://stahlke.org/dan/publications/qm652-project.pdf [Accessed 4 Jun. 2025].

4. A. Montina and Arecchi, F.T. (2008). Quantum Decoherence Reduction by Increasing the Thermal Bath Temperature. Physical Review Letters, [online] 100(12). doi:https://doi.org/10.1103/physrevlett.100.120401.

5. Jafari, R., Asadian, A., Abdi, M. and Akbari, A. (2025). Dynamics of decoherence in a noisy driven environment. *Scientific Reports*, [online] 15(1). doi:https://doi.org/10.1038/s41598-025-00815-8.

6. Joakim Bergli, Galperin, Y.M. and Altshuler, B.L. (2009). Decoherence in qubits due to low-frequency noise. *New Journal of Physics*, 11(2), pp.025002–025002. doi:https://doi.org/10.1088/1367-2630/11/2/025002.

8 Appendix A: Python Code for Rule Generation

```
i = 1

for x1 in range(1, 5):

        for x2 in range(1, 4):

                for x3 in range(1, 4):

                        G= (x1 + x2 + x3) / 3

                        print("Rule" + str(i) + " = ["+ str(x1), str(x2), str(x3), round(G), str(1), str(1)+"]")

                        i = i + 1
```

9 Appendix B: Theoretical Dataset

```
Theoretical_Data = [
    25.00, 0.50, 0.50;
    50.00, 0.70, 0.10;
    18.00, 4.00, 0.90;
    135.27, 9.48, 0.23;
    63.91, 2.15, 0.63;
    89.34, 7.73, 0.04;
    14.66, 0.18, 0.87;
    141.89, 3.02, 0.31;
    46.22, 5.57, 0.96;
    101.78, 1.13, 0.44
];
```